



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jakub Levý

Využití Puppeteeru pro automatizaci akcí webového prohlížeče

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Klímek, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V _____ dne _____

Podpis autora

Děkuji RNDr. Jakubu Klímkovi, Ph.D za příkladné vedení, věcné náměty
a připomínky k práci.

Název práce: Využití Puppeteeru pro automatizaci akcí webového prohlížeče

Autor: Jakub Levý

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Klímek, Ph.D.

Abstrakt: V této práci se zabýváme automatickým ovládáním webových prohlížečů. Naším cílem je vytvořit řešení umožňující zaznamenat akce, které bude možné uložit a opětovně vykonat. Vzhledem k tomu, že takových produktů existuje již řada, zejména pro ustálený framework Selenium, je pro nás zásadní využít novější a modernější knihovnu Puppeteer, pro kterou tyto nástroje neexistují. Práce začíná řešeršní částí některých knihoven a produktů, následně jsou stanoveny požadavky kladené na naše řešení, mezi které patří připojení ke vzdálené instanci prohlížeče a uživatelské rozhraní. Řešení bylo implementováno podle těchto požadavků. Součástí práce je zhodnocení a porovnání funkcionality oproti konkurenčním produktům. Krátce můžeme vyhodnotit – výhoda našeho řešení oproti produktům Selenium IDE a Katalon Recorder vytvořených nad Seleniem je možnost připojení na vzdálenou instanci prohlížeče. Nevýhodou je chybějící podpora podmínek a cyklů, které nebyly stanoveny v požadavcích, a proto nebyly implementovány.

Klíčová slova: Puppeteer, automatizace webového prohlížeče

Title: Usage of Puppeteer for automation of web browser actions

Author: Jakub Levý

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Klímek, Ph.D.

Abstract: In this thesis, we deal with the automatic control of web browsers. Our aim is to create a solution to record actions that can be saved and re-executed. As there is already a number of such products, especially for the established Selenium framework, it is crucial for us to use the newer and modern Puppeteer library for which these tools do not exist. The thesis begins with the research part of libraries and products, then the requirements for our solution are determined, which include remote browser connection and user interface. The solution was implemented according to these requirements. Part of the thesis involves evaluating and comparing functionality with competing products. We can briefly evaluate—the advantage of our solution over Selenium IDE and Katalon Recorder products built on Selenium is the ability to connect to a remote instance of a browser. The drawback is the lack of support for conditions and cycles that were not determined as a requirement, and, therefore, have not been implemented.

Keywords: Puppeteer, web browser automation

Obsah

Seznam nestandardních a méně známých zkratk	4
1 Úvod	5
1.1 Cíle práce	5
1.2 Struktura textu	5
1.3 Využitý software a technologie	6
2 Seznámení s technologiemi k ovládání prohlížečů	7
2.1 Selenium	7
2.2 Puppeteer	10
2.3 Playwright	13
2.4 PhantomJS	16
3 Automatizace prohlížeče	17
3.1 Konkurenční řešení	17
3.1.1 Selenium IDE	17
3.1.2 Katalon Recorder	22
3.1.3 Ranorex Recorder	24
3.1.4 Headless Recorder (dříve Puppeteer Recorder)	24
3.2 Využití Puppeteeru	27
3.2.1 WindowEvents	27
3.2.2 ViewportEvents	29
4 Řešení	35
4.1 Požadavky	35
4.1.1 Funkční	35
4.1.2 Nefunkční	36
4.2 Design	36
4.3 Scénáře použití	39
4.4 Porovnání funkcionality oproti konkurenčním řešením	42
4.5 Vývoj	47
4.6 Testování	47
4.6.1 Odhalené problémy	48
4.7 Administrátorská dokumentace	50
4.8 Tutoriály	51
4.8.1 Prvotní spuštění	51
4.8.2 Nová nahrávka	52
4.8.3 Připojení ke vzdálenému Chromu	59
4.8.4 Tlačítko JSON	61
4.8.5 Výběr akcí ke zpracování	62

4.8.6	Filtr akcí	64
4.8.7	Změna identifikátoru akce	67
4.9	Programátorská dokumentace	70
4.9.1	Backend	70
4.9.1.1	JsExtensions\ArrayExtensions.js	70
4.9.1.2	JsExtensions\ObjectExtensions.js	70
4.9.1.3	JsExtensions\StringExtensions.js	71
4.9.1.4	PuppeteerExtensions\BrowserExtensions.js	71
4.9.1.5	PuppeteerExtensions\PageExtensions.js	71
4.9.1.6	selenium-ide-src\LocatorPkg.js	72
4.9.1.7	WindowFunctions\LocatorTransformation.js	72
4.9.1.8	BrowserConnectionLayer.js	72
4.9.1.9	CodeGenerator.js	73
4.9.1.10	main.js	74
4.9.1.11	Optimizer.js	74
4.9.1.12	Recorder.js	75
4.9.1.13	WindowFunctionsUtils.js	75
4.9.1.14	ZeroMQUtils.js	75
4.9.2	Frontend	76
4.9.2.1	Forms\CodeGenEditor.cs	76
4.9.2.2	Forms\CodeGenSettingsForm.cs	77
4.9.2.3	Forms\FilterForm.cs	78
4.9.2.4	Forms\JsonEditor.cs	79
4.9.2.5	Forms\MainForm.cs	80
4.9.2.6	Forms\NodeJsConfig.cs	81
4.9.2.7	Forms\PlayerForm.cs	82
4.9.2.8	Forms\RecorderSettingsForm.cs	83
4.9.2.9	Forms\ReplayViewForm.cs	84
4.9.2.10	Forms\WaitingWindow.cs	84
4.9.2.11	UserControls\ActionUserControl.cs	85
4.9.2.12	UserControls\EditUserControl.cs	86
4.9.2.13	UserControls\PuppeteerOptionsUserControl.cs	88
4.9.2.14	UserControls\ThumbnailUserControl.cs	89
4.9.2.15	CodeGenOptions.cs	89
4.9.2.16	ConfigManager.cs	89
4.9.2.17	Configuration.cs	90
4.9.2.18	Constants.cs	90
4.9.2.19	CurrentEdit.cs	91
4.9.2.20	Filter.cs	91
4.9.2.21	NodeJsOptions.cs	91

4.9.2.22	PairSocketExtensions.cs	91
4.9.2.23	PlayerOptions.cs	92
4.9.2.24	Program.cs	92
4.9.2.25	PuppeteerOptions.cs	92
4.9.2.26	RecordedEvents.cs	93
4.9.2.27	RecorderConfiguration.cs	93
4.9.2.28	Recording.cs	93
4.9.2.29	RecordingManager.cs	93
4.9.2.30	Thumbnail.cs	94
4.9.2.31	ThumbnailManager.cs	94
4.9.2.32	WaitForNavigationEnum.cs	95
5	Závěr	96
A	Fragmenty kódu pro Puppeteer	97
A.1	Zachycení událost z viewportu	97
A.2	Momentálně aktivní tab	98
A.3	Oprávnění „push“	98
A.4	Element podle lokátoru	99
A.5	Úspěšné čtení ze schránky i zápis do schránky	99
A.6	Práce se schránkou	100
A.7	Pokus o získání oprávnění pro práci se schránkou	101
A.8	Čtení ze schránky s CDP oprávněním „clipboardReadWrite“	102
A.9	Využití NetMQ a ZeroMQ.js	103
	Slovníček vybraných pojmů	103
	Seznam obrázků	107
	Seznam tabulek	108
	Seznam ukázek kódu	109
	Reference	110

Seznam nestandardních a méně známých zkratk

CDP Chrome DevTools Protocol

Chrome Chromium/Google Chrome/Microsoft Edge

FRP Firefox Remote Protocol

Jazyk Programovací jazyk

Prohlížeč Webový prohlížeč

Selektor CSS Selektor

Stránka Webová stránka

1 Úvod

V dnešní době se potýkáme s trendem přesunu vývoje uživatelských rozhraní. Místo klasických aplikací se vyvíjí aplikace webové, jejichž hlavní výhodou je kompatibilita – fungují na všech platformách, na kterých je k dispozici webový prohlížeč. Tento trend přispívá k rychlejšímu vývoji, protože ubyla potřeba vyvíjet a udržovat více různých verzí stejné aplikace pro různé platformy.

Problém se však objevil v samotném automatickém testování aplikací, které není možné provést rychle a spolehlivě automaticky kvůli povaze webu, bez patřičných odborných znalostí je to téměř nemožné. Pro tyto účely existuje mnoho produktů, z nichž většina je postavena nad Seleniem (2.1).

V nedávné době se ale začaly objevovat další konkurenční knihovny a technologie: Puppeteer (2.2) a Playwright (2.3), které zatím nejsou rozšířené a neexistují pro ně nástroje na automatizaci webu.

Předtím než se čtenář přesune dále, dovolíme si upozornit na Slovníček vybraných pojmů, který se nachází na konci této práce. Uvedeny v něm jsou pouze pojmy, které jsou významné pro tuto práci a považujeme je za méně známé až nestandardní.

- API (Application Programming Interface) považujeme za známý pojem, ve slovníčku není uveden
- CDP (Chrome Devtools Protocol) považujeme za neznámý pojem, ale v textu je několikrát zmíněn, ve slovníčku je uveden

1.1 Cíle práce

Cílem práce je analyzovat, navrhnout, implementovat, zdokumentovat a otestovat software pro automatizaci akcí prováděných ve webovém prohlížeči Chrome pomocí existující knihovny Puppeteer [50]. Tato moderní knihovna pro JavaScript s běhovým prostředím Node.js disponuje vysokoúrovňovým API implementujícím podmnožinu funkcí CDP [14]. Řešitel se seznámí s dalšími knihovnami a technologiemi pro ovládání webových prohlížečů a srovná je s možnostmi Puppeteeru. Součástí práce bude rešerše alternativních řešení umožňujících automatizaci a jejich porovnání oproti vlastnímu navrženému řešení.

1.2 Struktura textu

Kapitola 2 obsahuje informace o knihovnách a technologiích běžně používaných k ovládání prohlížečů. Popíšeme zde pro nás zásadní Puppeteer, ale podrobněji rozvedeme i varianty (podle nás) běžnější, jimiž se budeme později inspirovat.

V kapitole 3 se nejprve přesuneme k produktům postaveným nad nástroji popsány v kapitole 2. S těmito produkty se seznámíme a popíšeme si jejich specifiky. Nejzásadnější produkt pro nás bude Headless Recorder, protože je jako jediný postavený nad Puppeteerem. Další významnou částí této kapitoly je rozbor možností Puppeteeru pro nahrávání a přehrávání akcí. Její součástí jsou kromě popisu i ukázky kódu demonstrující nahrávání a přehrávání jednotlivých akcí.

Nejdůležitější částí této práce je kapitola 4. Zde se zabýváme vlastním řešením, nejprve stanovíme požadavky kladené na naše řešení, které jsou získané na základě zkušeností z kapitol 2 a 3. Následují diagramy zobrazující možnosti uživatelů při práci s řešením, nechybí ani komplexní porovnání oproti řešením představeným v kapitole 3. Část Vývoj je zaměřená na nástroje použité během vývoje, dále následuje Testování, kde se dozvíme, jak bylo řešení testováno a jaké problémy byly při testování odhaleny. Na konci této kapitoly samozřejmě nechybí ani dokumentace pro administrátora, uživatele a programátora.

Poslední číslovaná kapitola této práce je kapitola 5, její hlavní přínos je zhodnocení splnění požadavků definovaných v kapitole 4.

Za zmínku stojí ještě appendix A, který obsahuje fragmenty kódu pro Puppeteer, které jsou příliš dlouhé nebo nejsou nutné pro pochopení textu. Na tyto fragmenty budeme v textu několikrát odkazovat, slouží zejména jako důkaz pro ověření některých tvrzení, které vyslovíme.

1.3 Využitý software a technologie

Vzhledem k tomu, že v této práci se zabýváme různými existujícími softwary a technologiemi, uvádíme zde jejich seznam, včetně konkrétně využitých verzí.

Název	Verze
Headless Recorder	0.8.1
Katalon Recorder	5.3.22
Node.js	15.5.0
PhantomJS	2.1.1
Playwright	1.3.0
Puppeteer	5.1.0
Ranorex Recorder	9.3.4
Selenium IDE	3.17.0

Tabulka 1: Využitý software a technologie včetně verzí

2 Seznámení s technologiemi k ovládání prohlížečů

V této části popíšeme a částečně porovnáme vybrané technologie k ovládání prohlížečů. Význam této kapitoly je ryze informativní, čtenářům znalých těchto technologií je doporučeno přečtení alespoň podsekcce 2.2, která popisuje knihovnu Puppeteer – zásadní stavební blok této práce.

2.1 Selenium

Selenium je sada nástrojů pro automatizaci prohlížečů. Jeho historie sahá až do roku 2004, kdy Jason Huggins vytvořil „JavascriptTestRunner“ pro testování webových aplikací v Pythonu. Během dalšího vývoje byl „JavascriptTestRunner“ přejmenován na Selenium (česky selen), který snižuje toxicitu rtuti. Nové jméno mělo zesměšnit tehdejší konkurenční společnost Mercury (česky rtuť). Později vznikl celý ekosystém Selenia, který zahrnuje browser drivers, selenium drivers, language bindings a testing frameworks [5, 61].

Browser driver

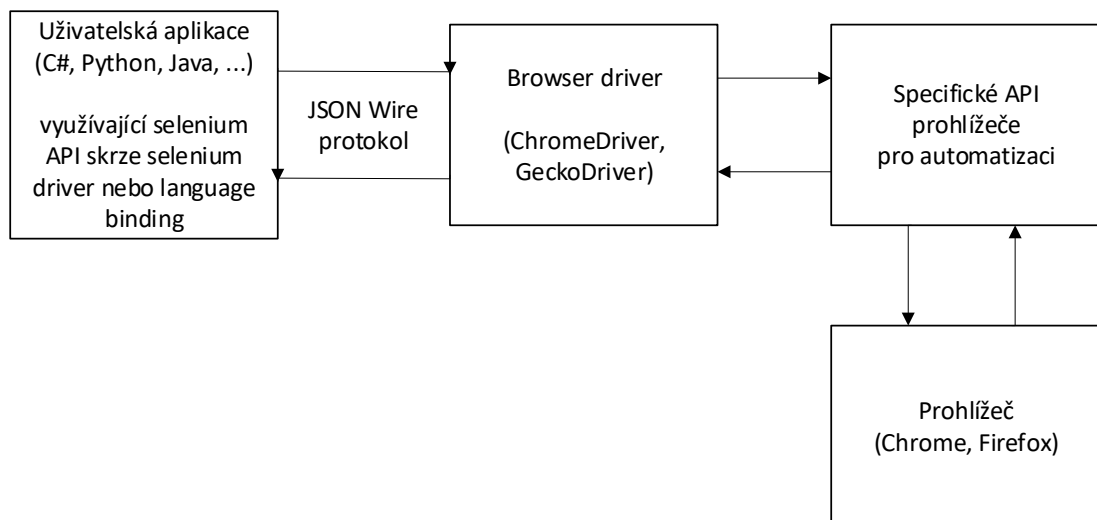
Je program vytvořený pro konkrétní prohlížeč, jehož specifické API je využíváno pro automatické ovládání. Driver dostává požadavky od uživatelské aplikace komunikující s driverem [17, 67].

Selenium driver

Jedná se o knihovnu pro určitý jazyk, která umožňuje komunikovat s browser driverem nativně pomocí selenium API přes JSON Wire protokol. [17, 67].

Language bindings

Jde se o specifický, tzv. slepovací kód, který umožňuje využívat selenium API i z jazyka, který není nativně podporován a neexistuje pro něj selenium driver [17, 67].



Obrázek 1: Komunikace Selenia s prohlížečem

Přesuneme se ke způsobu identifikace elementů na stránce. Dnes jsou v selektorech verze 3 definovány rozmanitější selektory např. E~F, který vybere element F, kterému předchází element E [59]. Stále však není možné napsat selektor podle vlastnosti `textContent` elementu [60]. Může se to sice zdát zprvu nedůležité, avšak často se jedná o dobrou identifikaci tlačítka, která je jednoznačná a uživatelsky přívětivá.

```
1 <button class="h51n1h-0" type="submit">Přihlásit se</button>
```

Ukázka kódu 1: HTML kód tlačítka pro přihlášení

Předpokládejme, že tlačítko obsahuje jednoznačný text uvnitř tagů, pak bychom ho mohli identifikovat textem „Přihlásit se“, místo třídy, která může být generována na session.

Problém identifikace elementů byl v Seleniu vyřešen pomocí rozšíření selektorů na tzv. lokátory [67].

Lokátor	Identifikuje element
class name	jehož třída obsahuje hledanou hodnotu.
css selector	odpovídající hledanému selektoru.
id	jehož atribut id odpovídá hledané hodnotě.
name	jehož atribut name je odpovídající hledané hodnotě.
link text	jehož viditelný text odpovídá hledané hodnotě.
partial link text	jehož viditelný text obsahuje hledanou hodnotu.
tag name	jehož název tagu odpovídá hledané hodnotě.
xpath	odpovídající XPath dotazu.

Tabulka 2: Seznam lokátorů podporovaných Seleniem [67]

Pomocí lokátorů bychom mohli identifikovat tlačítko z uk. k. 1. Vhodným lokátorem je `link text=„Přihlásit se“`. Nejuniverzálnější lokátor je ale XPath, s jehož využitím je možné přepsat všechny ostatní lokátory.

Mezi výhody Selenia patří podpora mnoha jazyků, mezi kterými nechybí i specifitější jazyky např. Dart, Haskell a R. Další výhodou je dobrá podpora napříč spektrem prohlížečů, obvykle včetně podpory všech OS, na kterých daný jazyk a prohlížeč běží [17, 67].

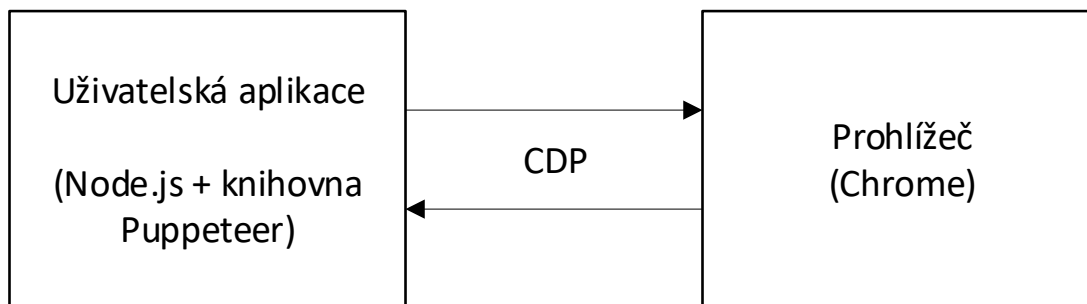
Prohlížeč	Podporované OS	Spravuje
Chromium/Chrome	Windows/macOS/Linux	Google
Firefox	Windows/macOS/Linux	Mozilla
Edge	Windows 10	Microsoft
Internet Explorer	Windows	Selenium Project
Safari	macOS El Capitan a novější	Apple
Opera	Windows/macOS/Linux	Opera

Tabulka 3: Prohlížeče podporované Seleniem [67]

Hlavní nevýhodou, částečně vyplývající ze zmíněných výhod, je omezená funkcionality selenia API podporující pouze to, co umí všechny prohlížeče, resp. všechny browser drivery.

2.2 Puppeteer

Puppeteer je moderní Node.js knihovna, její počátky sahají pouze do roku 2017, kdy byla vydána první verze [55]. Puppeteer poskytuje API pro ovládání Chromu¹, narozdíl od Selenia s prohlížečem komunikuje pomocí Chrome DevTools Protocol (CDP). To hned nabízí jednu výhodu a nevýhodu. Onou výhodou je podpora specifických a proprietárních funkcí Chromu. Nevýhodou je, že Chrome je jediným podporovaným prohlížečem [49].



Obrázek 2: Komunikace Puppeteeru s Chromem

Puppeteer nelze považovat za konkurenci Selenia. Selenium se zaměřuje na automatizaci nezávisle na prohlížeči. Jeho hodnotou je standardní API, které funguje ve všech hlavních prohlížečích. Puppeteer se narozdíl od toho zaměřuje na Chrome. Hodnotou Puppeteeru je bohatší funkcionalita a jednodušší API [49].

Pro každou verzi Puppeteeru existuje právě jedna verze Chromu, se kterou je garantována funkčnost.

Verze Chromu	Verze Puppeteeru
86.0.4240.0	5.3.0
85.0.4182.0	5.2.1
84.0.4147.0	5.1.0
83.0.4103.0	3.1.0

Tabulka 4: Několik odpovídajících verzí Chromu a Puppeteeru [50]

Z tohoto důvodu existují dva balíčky. Prvním je balíček „puppeteer“, který obsahuje kromě knihovny i vestavěný Chrome kompatibilní verze. Druhý balíček „puppeteer-core“ obsahuje pouze knihovnu. „puppeteer-core“ se může hodit pokud si chceme Chrome spravovat sami nebo se Puppeteerem chceme připojit na vzdálený Chrome na jiném počítači a vestavěný by nám tak zabíral zbytečně místo [49].

¹ Aktuálně již existují nightly verze Firefoxu s FRP, jehož implementace je založená na CDP. Platí, že $FRP \subset CDP$. Je možné použít Puppeteer s Firefoxem, ačkoliv se zatím jedná pouze o experimentální podporu [56, 57].

Podíváme se na několik ukázek porovnání Node.js kódu Puppeteeru a Selenia. První ukázkou bude vytvoření screenshotu stránky seznam.cz.

```
1 const fs = require('fs')
2 const {Builder} = require('selenium-webdriver')
3 const chrome = require('selenium-webdriver/chrome');
4
5 (async () => {
6   let driver = await new Builder().forBrowser('chrome')
7     .setChromeOptions(new chrome.Options().headless())
8     .build()
9
10  await driver.get('http://seznam.cz')
11  const imageData = await driver.takeScreenshot()
12  fs.writeFileSync('C:\\scrnshot.png', imageData, 'base64')
13  await driver.quit()
14 }) ()
```

(a) Node.js Selenium

```
1 const puppeteer = require('puppeteer');
2
3 (async () => {
4   const browser = await puppeteer.launch()
5   const page = (await browser.pages())[0]
6   await page.goto('http://seznam.cz')
7   await page.screenshot({path: 'C:\\scrnshot.png', fullPage: true})
8   await browser.close()
9 }) ()
```

(b) Puppeteer

Ukázka kódu 2: Screenshot v Node.js kódu

Kód (a) i (b) spustí Chrome v headless režimu, načte seznam.cz a uloží screenshot do souboru C:\scrnshot.png. Jediný rozdíl je ten, že kód (b) provede screenshot celé stránky, jak je nastaveno argumentem `fullPage: true`. Selenium API narozdíl od Puppeteeru neumožňuje provést screenshot celé stránky [37] .

Druhá ukázka nechá vyhodnotit JavaScript uvnitř stránky prohlížeče a výsledek vypíše na výstup. Pro ušetření místa vynecháme v této a v následujících ukázkách této podsekcce nedůležitý kód, zejména kód inicializující a uklízející.


```
1 const name = await driver.executeScript('return navigator.appName')
2 console.log(name)
```

(a) Node.js Selenium

```
1 const name = await page.evaluate(() => navigator.appName)
2 console.log(name)
```

(b) Puppeteer

Ukázka kódu 3: Node.js kód pro vyhodnocení JavaScriptu prohlížeče

V tomto případě si můžeme všimnout, že se kód pro Selenium a Puppeteer příliš neliší. Oba fragmenty kódu vypíší textový řetězec „Netscape“, protože hodnota `navigator.appName` je takto definována [44].

Existují však i situace, kdy je jednodušší napsat kód pro Selenium, než pro Puppeteer, viz následující ukázka.

```
1 await driver.findElement(By.linkText('About')).then(e => e.click())
2 await driver.findElement(By.name('options')).then(e => e.click())
```

(a) Node.js Selenium

```
1 await page.$x("//*[text()='Grading']").then(es => es[0].click())
2 await page.$x("//*[@name='options']").then(es => es[0].click())
```

(b) Puppeteer

Ukázka kódu 4: Node.js kód pro kliknutí na element

První řádek i druhý řádek vždy obsahují kód, který nalezne a stiskne element. V případě prvního řádku se stiskne element s textovým popiskem „About“. Na druhém řádku se stiskne element obsahující atribut `name` s hodnotou „options“. Puppeteer narozdíl od Selenia nemá lokátory, elementy můžeme hledat pouze pomocí selektoru a naštěstí i XPath [50].

2.3 Playwright

Playwright je ryze moderní záležitostí vyvíjenou společností Microsoft vydanou teprve v druhém měsíci roku 2020 [54]. Projekt začal jako fork knihovny Puppeteer. Playwright však dává důraz na jiné aspekty funkcionality [27].

Rozdíly proti Puppeteeru

Playwright nabízí jednotné API pro automatizaci Chromu, Firefoxu a prohlížečů postavených na jádře WebKit [46]. Aktuálně je však bez úprav podporovaný pouze Chrome. Ostatní prohlížeče musí být speciálně upraveny pomocí patchů, aby bylo možné jejich použití s Playwrightem. Pro Playwright by bylo ideální sloučení těchto patchů se zdrojovými kódy prohlížečů, v budoucnu by k tomu mohlo dojít. Některé patche již byly přijaty. [48].

Další změnou je automatické čekání na elementy, před provedením akce s nimi [46]. První ukázkou kódu pro Playwright uvedeme korektně včetně inicializace a uklízení. Všechny další ukázky této podsekcce budou obsahovat pouze nezbytný kód.

```
1 const { chromium } = require('playwright');
2
3 (async () => {
4   const browser = await chromium.launch({headless: false})
5   const page = await browser.newPage()
6   await page.goto('https://webik.ms.mff.cuni.cz')
7   await page.click('#navlink_tab_grading')
8   await browser.close()
9 }) ()
```

(a) Korektní kód pro Playwright

```
1 await page.goto('https://webik.ms.mff.cuni.cz')
2 await page.click('#navlink_tab_grading')
```

(b) Nekorektní kód pro Puppeteer

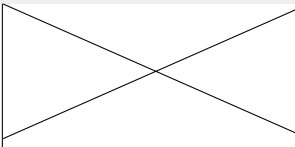
```
1 await page.goto('https://webik.ms.mff.cuni.cz')
2 await page.waitForSelector('#navlink_tab_grading')
3 await page.click('#navlink_tab_grading')
```

(c) Korektní kód pro Puppeteer

Ukázka kódu 5: Porovnání kódu Playwrightu a Puppeteeru pro stisk elementu po navigaci

Kód (a) na 7. řádku před klikem počká až element bude dostupný na stránce [47]. Naproti tomu (b) na element nečeká a pokud se nevyskytuje na stránce, tak vyhodí okamžitě výjimku [50]. Vzhledem k povaze dnešního webu je žádoucí si zkontrolovat a případně počkat na dynamické načtení elementů pomocí JavaScriptu. V Puppeteeru to můžeme zajistit viz 2. řádek (c) [50]. Povšimněme si, že speciálně kód na 1. řádku (c) nevyžaduje obdobné volání, které by počkalo na dokončení navigace. Puppeteer totiž automaticky čeká na volání „load“ události [79] uvnitř stránky [50].

Chování Playwrightu a Puppeteeru při navigaci a libovolné akci s elementem nám shrnuje následující tabulka.

Knihovna	Akce s elementem	Navigace	Vynutit čekání
Playwright	Čeká 30s, pokud se neobjeví, tak výjimka.	Čeká max 30s na zavolání „load“ události, jinak výjimka.	
Puppeteer	Pokud neexistuje, tak výjimka.		Voláním odpovídající „waitFor...“ metody.

Tabulka 5: Porovnání čekání Playwrightu a Puppeteeru [47, 50]

Jak již bylo zmíněno ve 2.2 na straně 12 v Puppeteeru je možné identifikovat element pouze pomocí selektoru a XPath, což sice nesnižuje sílu pro identifikaci, ale nutí nás psát si vlastní nestandardní rozšíření, nebo stále dokola psát XPath dotazy. Mezi další funkce Playwrightu patří, obdobně jako lokátory v Seleniu, sofistikovanější prostředky pro identifikaci elementů [47].

```

1 await page.click('text="banán"')
2 await page.type('[data-purpose="amount"]', '42')
3 await page.check(
4   'css=form[method = "POST"] >> css=input[type = "checkbox"]')
```

Ukázka kódu 6: Akce se specificky identifikovanými elementy

Na 1. řádku se provede stisknutí elementu s textovým popiskem „banán“. Další řádek „napíše“ hodnotu 42 do elementu s atributem data-purpose = "amount". Poslední nejdelší řádek obsahuje speciální binární, zleva asociativní operátor >>. Právý operand se vyhodnotí vzhledem k elementům splňujícím levý operand [47]. V tomto případě se hledá checkbox pro zaškrtnutí pouze uvnitř formuláře odesílaného POST metodou. Ekvivalentně bychom mohli použít XPath:

```

1 //form[@method="POST"]//input[@type="checkbox"]
```

Ukázka kódu 7: XPath ekvivalentní identifikátoru elementu na 4. řádku

Unikátní a zajímavá funkce, kterou disponuje pouze Playwright je API pro stahování souborů. Použití je následující [47]:

1. Stránka, potažmo kontext¹ prohlížeče, ve kterém stránka běží, musí mít nastavený parametr `acceptDownloads: true`.
2. Při provedení akce, která způsobí stažení souboru musíme počkat na událost „download“ a zachytit zachytit objekt `Download`.
3. Nyní můžeme využít API objektu `Download` a stažený soubor např. uložit do libovolného adresáře.

Uvedeme jednoduchý příklad, který stáhne instalátor Firefoxu a uloží ho na disk C s původním názvem.

```
1 const page = await browser.newPage({acceptDownloads: true})
2 await page.goto('https://www.mozilla.org/cs/firefox')
3
4 const [download] = await Promise.all([
5   page.waitForEvent('download'),
6   page.click('[data-download-os="Desktop"]')
7 ])
8
9 await download.saveAs('C:\\' + download.suggestedFilename())
```

Ukázka kódu 8: Stažení instalátoru Firefoxu

¹ Prostředí jedné i více stránek prohlížeče. Obvykle se sdíleným nastavením, např. neukládat historii. Při používání Chromu můžeme vytvořit nový kontext otevřením prvního okna anonymního režimu, zkratka CTRL+Shift+N. Pokud takto otevřeme další okna, nevytvoří se nový anonymní kontext, pouze se nová stránka přidá do existujícího anonymního kontextu.

2.4 PhantomJS

Podíváme se ještě krátce na značně odlišnou záležitost, která není od března roku 2018 aktivně vyvíjena [7]. Zatím jsme v kapitole 2 rozebírali jednotlivé technologie pro automatizaci existujících standardních prohlížečů. PhantomJS je ale kompletní prohlížeč navržený tak, aby byl skriptovatelný. Jednou z jeho předností ve své době bylo headless spouštění [45], které se ve Chromu objevilo až s verzí 59 pro macOS a Linux, podpora Windowsu pak byla přidána ve verzi 60 [22]. Firefox jednotně podporoval headless režim na macOS/Linuxu/Windowsu od verze 56 [71].

Prohlížeče	Datum vydání
Chrome 59	červen 2017
Chrome 60	červenec 2017
Firefox 56	listopad 2017
PhantomJS 1.0.0	leden 2011

Tabulka 6: Porovnání data vydání prvních verzí prohlížečů podporujících headless režim [20, 31, 52]

Z tabulky je patrné, že PhantomJS si držel konkurenční výhodu dlouhých 6 let. Po její ztrátě netrvalo dlouho a projekt byl archivován.

Na závěr si ukážeme fragment kódu využívající Node.js binding pro PhantomJS. Provedeme screenshot celé stránky `mff.cuni.cz` a uložíme ho jako `C:\scrnshot.pdf`.

```
1 const phantom = require('phantom');
2
3 (async () => {
4   const browser = await phantom.create()
5   const page = await browser.createPage()
6   await page.open('https://mff.cuni.cz')
7   await page.render('C:\\scrnshot.pdf')
8   await browser.exit()
9 }) ()
```

Ukázka kódu 9: Uložení screenshotu celé stránky do PDF s využitím PhantomJS

Obdobně jednoduchým způsobem je možné vytvořit screenshot s uložením do PDF i s použitím Puppeteeru nebo Playwrightu. V případě použití Selenia je nutné se omezit pouze na výstup v obrázkovém formátu [67] a na screenshot viewportu, jak již bylo zmíněno ve 2.2 Puppeteer na straně 11.

3 Automatizace prohlížeče

V této kapitole se podíváme na řešení k zaznamenávání uživatelských akcí v prohlížeči s možností následného opětovného vykonání, dále pouze uváděno jako řešení k nahrávání a přehrávání akcí, nebo pouze řešení.

Nejprve se podíváme na vybraná existující konkurenční řešení. Podsekce 3.1 Konkurenční řešení slouží k představení běžných zástupců řešení k automatizaci prohlížeče, zde jsou uvedeni zástupci s open-source i proprietárními licencemi. Hlavním významem této podsekce je ukázat existenci/neexistenci funkcí představených řešení. Tyto řešení jsou pak porovnány s vlastním řešením ve 4.4.

Později ve 3.2 se zaměříme výhradně na Puppeteer, zde provedeme detailní rozbor možností Puppeteeru pro nahrávání a přehrávání akcí včetně ukázek kódu demonstrujících využití.

3.1 Konkurenční řešení

V této části se zaměříme výhradně na konkurenční řešení, podíváme se na jejich funkcionalitu a popíšeme výhody a nevýhody.

Přikládáme ještě tabulku primárně určenou pro rychlou orientaci v této sekci a jednoduché porovnání.

Název	Využívá	Licence	Nahrávání	Přehrávání
Selenium IDE	Selenium	Apache License 2.0	✓	✓
Katalon Recorder	Selenium	Apache License 2.0	✓	✓
Ranorex Recorder	Proprietární		✓	✓
Headless Recorder	Puppeteer	Apache License 2.0	✓	✗

Tabulka 7: Orientační porovnání existujících řešení

3.1.1 Selenium IDE

Selenium IDE je nástroj postavený nad Seleniem viz 2.1 fungující jako extension Chromu a Firefoxu. Původně ho navrhl Shinya Kasatani, který v roce 2006 umožnil jeho vstup mezi oficiální nástroje Selenia [23, 62].

V roce 2017, kdy byla aktuální v2 Selenia IDE, se projekt dostal do problémů, kód v té době fungoval pouze s Firefoxem a byl závislý na specifickém API Firefoxu pro add-ony [4]. Těmto add-onům se dnes říká „legacy extensions“, ve Firefoxu byla jejich podpora úplně odstraněna s verzí 57 [21], která vyšla ve stejném roce [20]. Projekt se ale díky práci vývojářů podařilo udržet naživu.

Aktuálně existují dvě vyvíjené větve, první je v3, ve které se stále ještě přepisují některé části kódu starého Selenia IDE v2. Cílem v3, postaveného na extension API, je podpora aktuálních verzí prohlížečů Chrome a Firefox včetně všech funkcí původního Selenia IDE v2 [23].

Druhou větví je master, ve které se pracuje na přechodu z rozšíření Chromu a Firefoxu na Electron. V současné době zatím nebyla vydána žádná verze fungující s Electronem. Zájemci o vyzkoušení mají možnost si zdrojový kód zkompilovat sami [23].

Léta vývoje se projevují na funkcionalitě, pojďme se na některé zajímavé funkce současně nejnovějšího Selenia IDE v3.17.0 [53], podívat.

Výběr z lokátorů

Pro každý element je nalezena sada lokátorů, uživatel si může vybrat z nich ten nejvhodnější.

	Command	Target
1	open	/
2	set window size	1036x1004
3	click	id=navlink_tab_labs

Command

click

//

Target

id=navlink_tab_labs

Value

id=navlink_tab_labs

id

Description

linkText=Labs

linkText

css=#navlink_tab_labs

css:finder

xpath=//a[contains(text(), 'Labs')]

xpath:link

xpath=//a[@id='navlink_tab_labs']

xpath:attributes

xpath=//div[@id='header']/nav/a[4]

xpath:idRelative

xpath=//a[contains(@href, 'Labs')]

xpath:href

Obrázek 3: Výběr z lokátorů identifikujících element

Změna elementu klikem myši

Pro změnu elementu na němž se má akce provést, aniž bychom museli ručně přepisovat lokátor, existuje tlačítko umožňující znovu zachytit element.



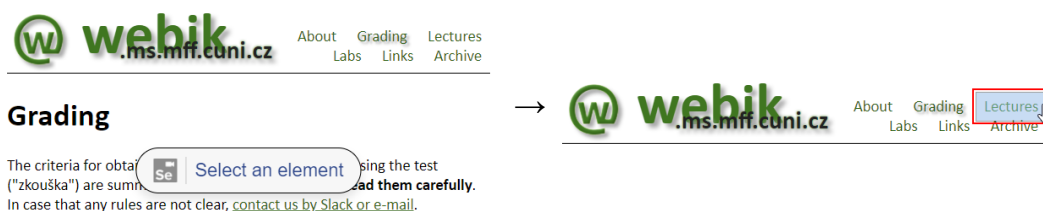
Command: click

Target: id=navlink_tab_labs

Value:

Description:

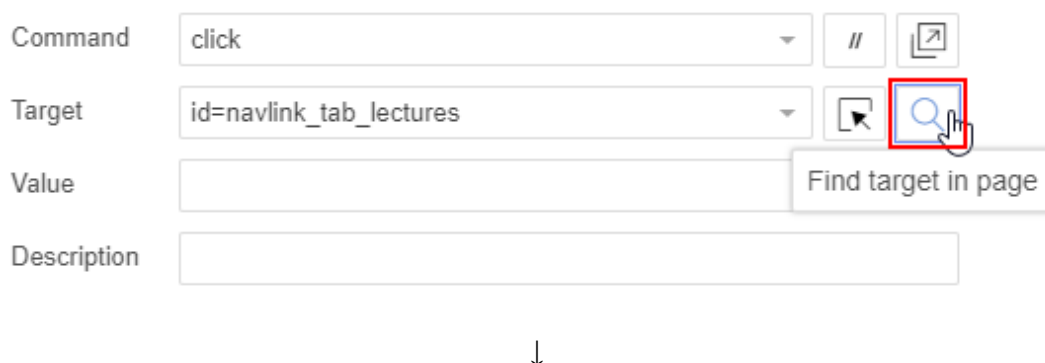
Select target in page



Obrázek 4: Změna elementu bez ručního přepisování lokátoru

Zobrazení nalezeného elementu

Pokud chceme zpětně zjistit jaký element je lokátorem identifikován, stačí kliknout na tlačítko s obrázkem lupy, identifikovaný element se uvnitř stránky zvýrazní.



Command: click

Target: id=navlink_tab_lectures

Value:

Description:

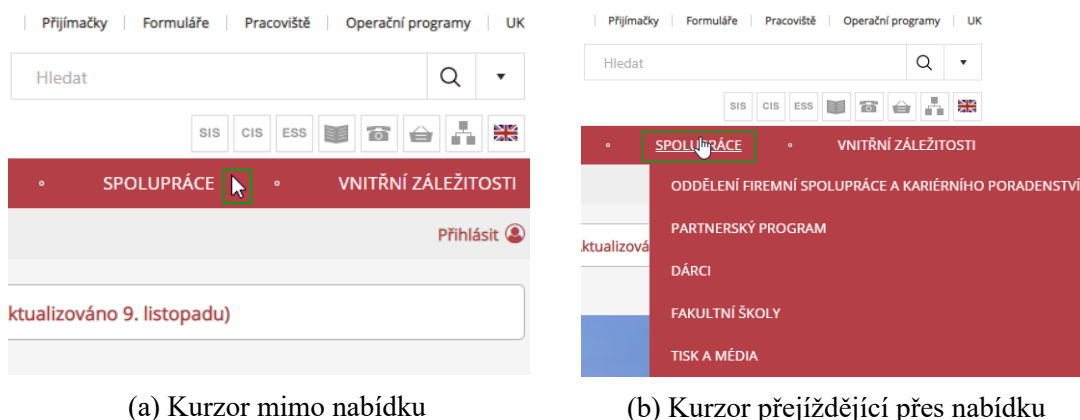
Find target in page



Obrázek 5: Zpětné zobrazení identifikovaného elementu

Mouseover

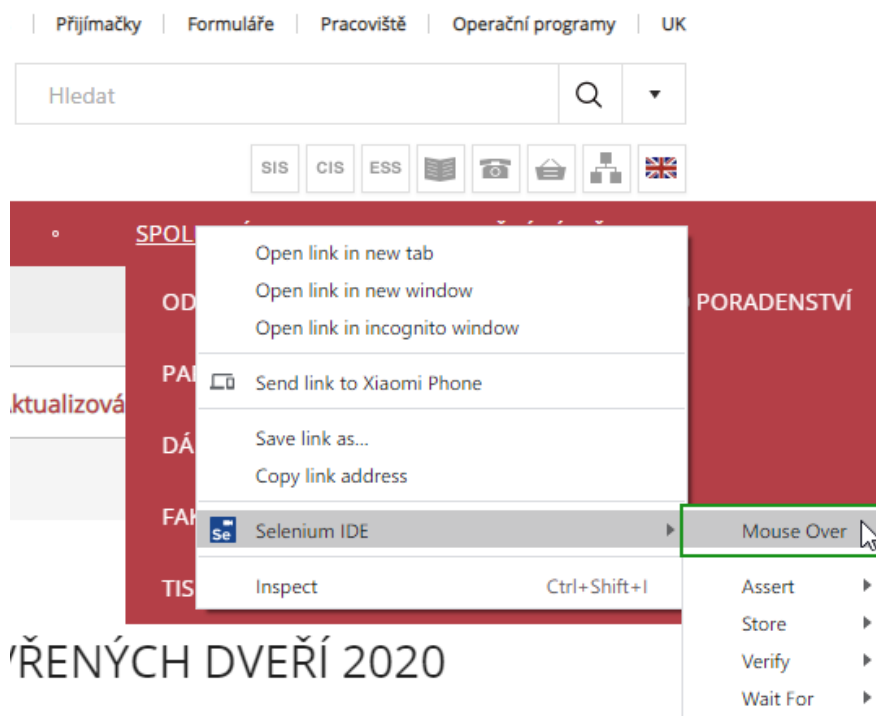
Mnoho elementů na stránce je často skryto dokud nepřejedeme myší přes nabídkový element, jehož JavaScript způsobí odkrytí elementů.



Obrázek 6: Stránky MFF UK s nabídkovým elementem

Není vhodné zaznamenávat veškeré pohyby kurzoru přes elementy. Většina z nich nezpůsobuje žádný efekt na stránce, pouze nám zahlcují nahrávač přebytečnými událostmi a snižují přehlednost. Jak ale nahrát nezbytná přejetí kurzoru přes element? Ukážeme si jak je tento problém vyřešen v Selenium IDE.

Implicitně se nezaznamenávají žádné „mouseover“ události. Zaznamenání si můžeme vyžádat explicitně.



Obrázek 7: Explicitní zaznamenání „mouseover“ akce

Debugger

Selenium IDE obsahuje v názvu zkratku IDE oprávněně. Jeho součástí je plnohodnotný debugger včetně breakpointů a krokování.

The screenshot displays the Selenium IDE interface. At the top, the project is named 'Nákup*' and the status is 'Paused in debugger'. The left sidebar shows the project name 'Pistácie+čokoláda*'. The main area shows a list of commands with a table structure:

	Command	Target	Value
1	mouse over	css=.hover > a > div	
2	click	linkText=Pistácie	
3	click	linkText=GRIZLY Pistácie loupané 1000 g	
4	click	xpath=//div[2]/button	
5	click	css=#sidebar-search-form-1 .text	
6	type	name=seastr	čokoláda
7	send keys	name=seastr	\${KEY_ENTER}

Below the table, the details for the selected command (3) are shown:

Command: click
Target: linkText=GRIZLY Pistácie loupané 1000 g
Value:
Description:

At the bottom, the 'Log' tab shows the execution history:

Log	Reference
Running 'Pistácie+čokoláda'	16:17:45
1. Trying to find css=.hover > a > div... OK	16:17:45
2. click on linkText=Pistácie OK	16:17:58
3. click on linkText=GRIZLY Pistácie loupané 1000 g	16:17:58

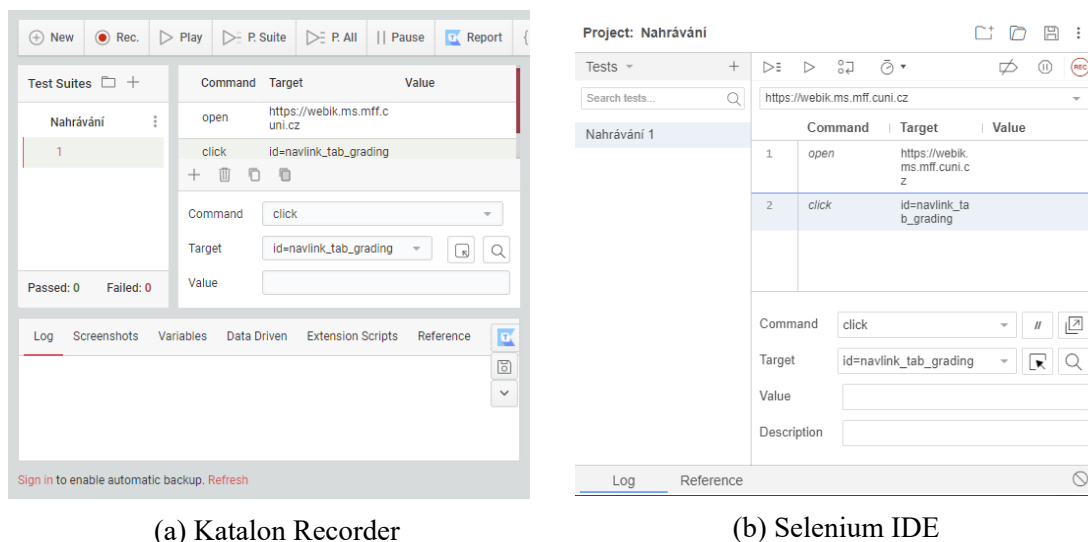
Obrázek 8: Přehrávání s breakpointem na 3. akci

Podívejme se na nevýhody aktuálního Selenia IDE v3, tou větší, než by se na první pohled mohlo zdát, je jeho návrh závislý na API prohlížečů pro extensiony. Připomeňme: v3 je dnes extension pro Chrome i Firefox, narozdíl od v2, která je závislá na specifickém API Firefoxu, fungující pouze s Firefoxem. Úplného osvobození od API prohlížečů bude dosaženo až s vydáním verze postavené na Electronu.

Kritickou záležitostí aktuálního Selenia IDE v3 je, že může komunikovat pouze s instancí prohlížeče, která ho spustila, tzn. že nemůže komunikovat s prohlížečem bez nainstalovaného extensionu, případně s prohlížečem běžícím na vzdáleném počítači.

3.1.2 Katalon Recorder

Katalon Recorder je produkt společnosti Katalon LLC, začal jako fork Selenia IDE, od vydání první veřejné verze v roce 2016 ale fungoval s Chromem a Firefoxem [41,43]. V roce 2017, kdy Selenium IDE bylo kompatibilní pouze s Firefoxem a využívalo jeho historické, nyní odstraněné API, byl Katalon Recorder velmi silný konkurent. Vzhledem k tomu, že se dnes Selenium IDE ze svých problémů dostalo, se jedná o dvě velmi podobná a srovnatelná řešení.

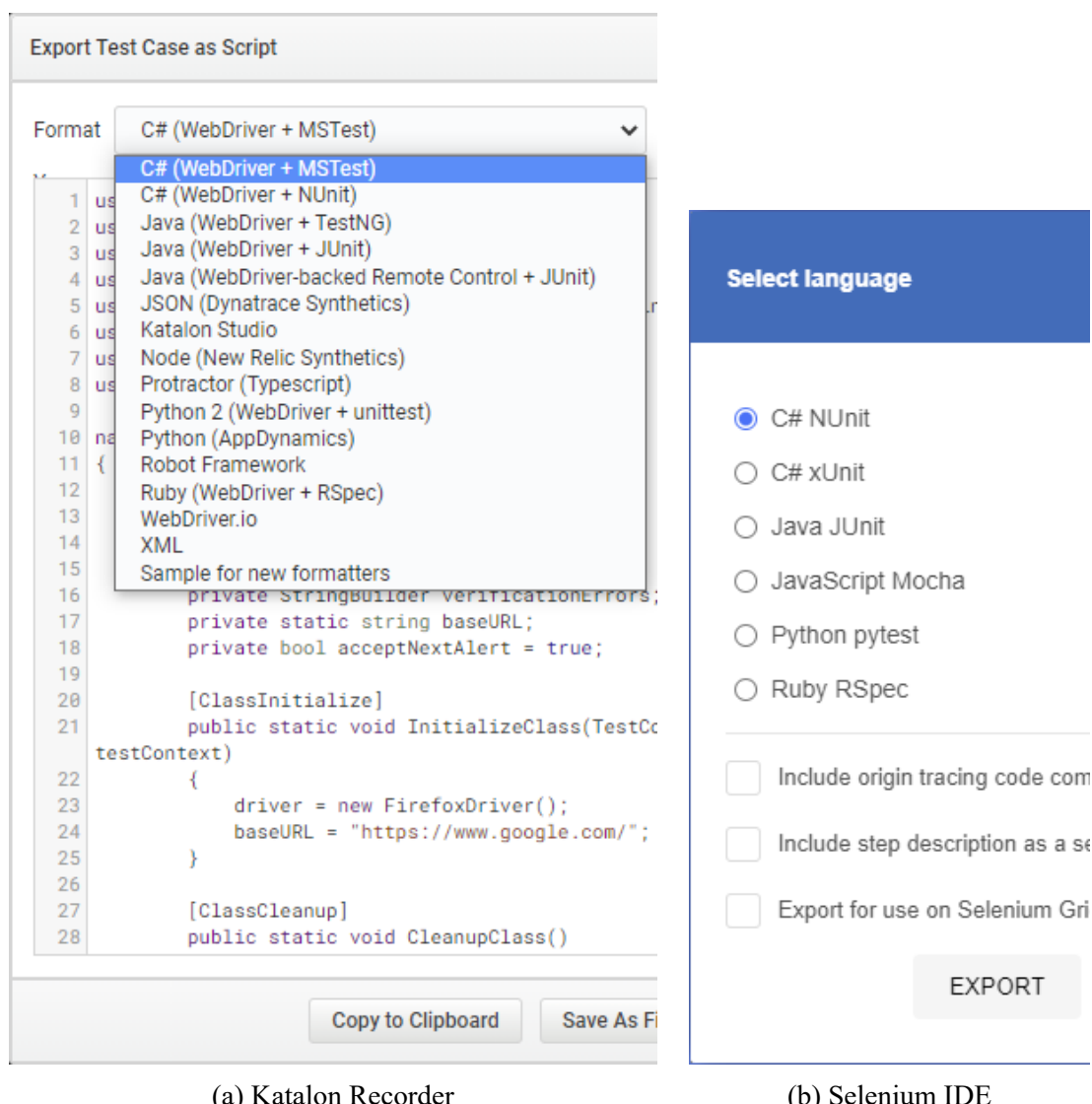


(a) Katalon Recorder

(b) Selenium IDE

Obrázek 9: Porovnání GUI Katalon Recorderu a Selenia IDE

Hlavním důvodem pro upřednostnění Katalon Recorderu při výběru řešení je jeho placená varianta Katalon Studio Enterprise, ve které je zahrnuta podpora produktu, ale i rozšířené funkce. Jednou takovou funkcí je „self-healing execution“, tj. využití nalezených náhradních lokátorů pokud vybraný selže [42]. Existují ale i další méně patrné funkce Katalon Recorderu zahrnuté i ve volně dostupné verzi, které Selenium IDE nemá.



(a) Katalon Recorder

(b) Selenium IDE

Obrázek 10: Porovnání podporovaných jazyků pro export

Jak obr. 10 ukazuje, Katalon Recorder dokáže narozdíl od Selenia IDE exportovat nahrané akce do většího počtu různých technologií.

Na druhou stranu v Katalon Recorderu jsou dostupné pouze tři lokátory a těmi jsou id, name a xpath [40] (seznam lokátorů viz tab. 2 na straně 9). Ostatní jsou zpřístupněny až v Katalon Studiu Enterprise a Katalon Studiu [40], což je bezplatná varianta Katalon Studia Enterprise postrádající mimo jiné podporu. Ani tuto variantu není možné stáhnout bez vytvoření Katalon účtu vyžadujícího jméno, email, heslo a zodpovězení dvou otázek [42].

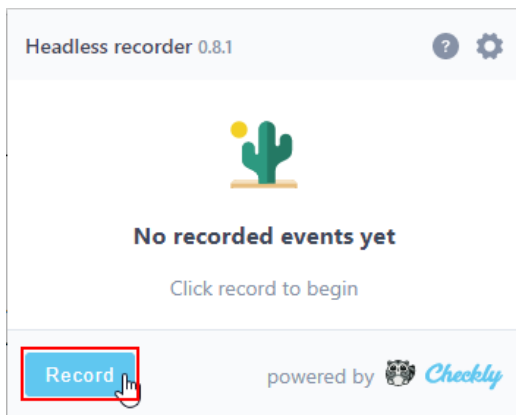
3.1.3 Ranorex Recorder

Velmi krátce zmíníme jedno proprietární komerční řešení, tím je Ranorex Recorder, který je součástí Ranorex Studia určeného pro automatizaci GUI a prohlížečů. Původně byl tento produkt vytvořen v Rakousku v roce 2007 jako volně dostupná aplikace. Její následná popularita vedla k vývoji komerční verze s podporou, která byla v roce 2017 odkoupena americkou společností IDERA, Inc [15].

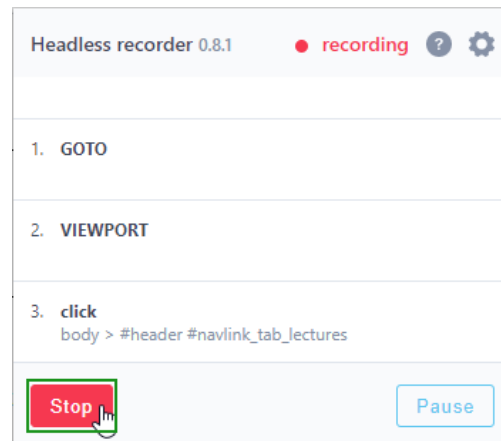
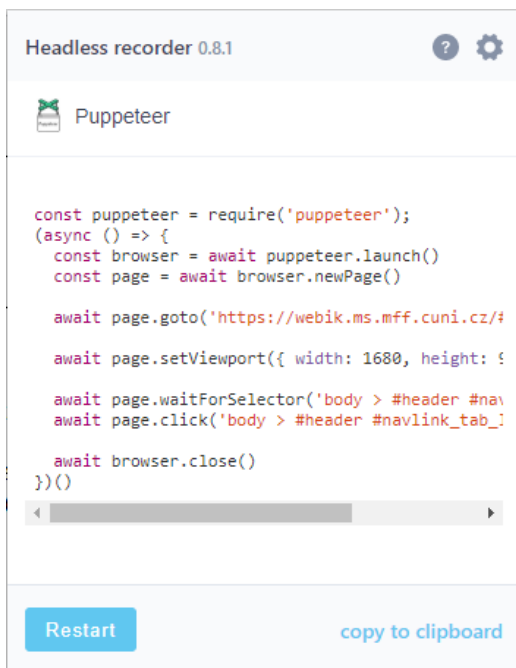
Pomineme GUI a zaměříme se pouze na automatizaci prohlížečů, tato funkčnost je zajištěna na bázi komunikace externí aplikace Ranorex Recorderu a prohlížeče, resp. jeho rozšíření Ranorex Automation. Mezi prohlížeče, pro které existuje rozšíření Ranorex Automation a jsou proto podporovány, patří Internet Explorer, Firefox a Chrome.

3.1.4 Headless Recorder (dříve Puppeteer Recorder)

Jediným řešením postaveným na Puppeteeru, o kterém se zmíníme je Headless Recorder od společnosti Checkly, Inc. Řešení je postavené nad Puppeteerem, a proto je kompatibilní pouze s Chromem. Celá implementace je provedena jako rozšíření prohlížeče, nejedná se o úplné řešení umožňující nahrávání a přehrávání akcí. Podporované je pouze zaznamenávání několika málo akcí a vygenerování Puppeteer kódu [24].



→ Zaznamujeme akce, jejichž kód chceme vygenerovat.



Obrázek 11: Použití Headless Recorderu

Funkcionalita

Mezi zaznamenávané akce patří základní události jako stisk klávesy a myši. Podpora „mouseover“ události v jakékoliv podobě chybí stejně jako podpora lokátorů či případných jiných rozšíření selektorů.

V nastavení je k dispozici, jak ukazuje následující obrázek, pouze několik položek, jejichž význam je zřejmý.

Headless Recorder Options

Code Recorder settings

CUSTOM DATA ATTRIBUTE

Define an attribute that we'll attempt to use when selecting the elements, i.e. "data-custom". This is handy when React or Vue based apps generate random class names.


Code Generator settings

- ☒ Wrap code in async function
- ☒ Set headless in puppeteer launch options
- ☒ Add `waitForNavigation` lines on navigation
- ☒ Add `waitForSelector` lines before every `page.click()`
- ☒ Add blank lines between code blocks

Extension settings

- ☐ Allow recording of usage telemetry

We only record clicks for basic product development, no website content or input data. Data is never, ever shared with 3rd parties.

sponsored by  Checkly

Obrázek 12: Nastavení Headless Recorderu

Právě jedním ze zásadních důvodů pro vytvoření této práce byly chybějící pokročilé funkce jako např. nemožnost přehrát zaznamenané akce.

3.2 Využití Puppeteeru

Uvědomme si, že Puppeteer je určen pro automatizaci prohlížeče. Naším cílem je využít ho k nahrávání a přehrávání akcí s možností zobrazení kódu vygenerovaného pro Puppeteer. Z důvodů tohoto neobvyklého použití se objevilo několik problémů, které ukážeme spolu s popisem využití Puppeteeru pro tento účel.

Podíváme se na jednotlivé akce s Chromem, rozdělíme si je na dvě skupiny. První bude zahrnovat akce s oknem prohlížeče, např. otevření tabu, zavření tabu, přepnutí aktivního tabu apod. Tyto akce nazveme „WindowEvents“. Druhou skupinou zvanou „ViewportEvents“ budeme rozumět akce odehrávající uvnitř viewportu prohlížeče, např. kliknutí na element, rolování stránky, odeslání formuláře apod.

V dalších částech budeme demonstrovat způsoby zachycení a kód k vygenerování pro opětovné přehrávání akcí. Ukázky kódu budou postrádat nezajímavé rušivé části jako inicializace apod. Pokud by byl kód příliš dlouhý nebo ho nepovažujeme za nezbytný, budeme odkazovat do appendixu A, kde jsou všechny takové ukázky umístěny.

3.2.1 WindowEvents

Začneme s přímočarými událostmi, pro které je v Puppeteeru připraveno dobře fungující API. Poté se přesuneme k akcím, které se nedají zachytit triviálním způsobem nebo mají složitý kód pro opětovné přehrávání.

Otevření nového tabu

```
1 browser.on('targetcreated', target => {  
2   if (target.type() === 'page')  
3     console.log('Vytvořen nový tab s url: ' + target.url())  
4 })
```

Ukázka kódu 10: Zachycení otevření nového tabu

Vzhledem k tomu, že typ proměnné `target` nemusí být vždy „page“, jež odpovídá otevření nového tabu, je podmínka na 2. řádku potřebná.

Obdobně jednoduchý je i kód pro přehrávání. Malým nedostatkem je, že API Puppeteeru neumožňuje otevřít nový tab s konkrétní adresou URL, proto musíme hned po otevření adresu změnit.

```
1 const newPage = await browser.newPage()  
2 await newPage.goto(url)
```

Ukázka kódu 11: Otevření nového tabu

Uzavření existujícího tabu

```
1 browser.on('targetdestroyed', target => {
2   if (target.type() === 'page')
3     console.log('Uzavřen existující tab s url: ' + target.url())
4 })
```

Ukázka kódu 12: Zachycení uzavření existujícího tabu

```
1 await page.close()
```

Ukázka kódu 13: Uzavření existujícího tabu

Kód uk. k. 13 uzavře objekt `page`, kterému odpovídá nějaký tab. Abychom věděli, který tab máme uzavřít, potřebujeme jeho jednoznačnou identifikaci. API Puppeteeru žádnou takovou identifikaci nedisponuje. Jedinou naší možností je vytvoření vlastního unikátního identifikátoru. Abychom měli jistotu, že každý tab obsahuje identifikátor, nejlepší je přidělit mu ho hned po vytvoření tabu, např. v události `targetcreated`.

Změna adresy URL

```
1 browser.on('targetchanged', async target => {
2   if (target.type() === 'page') {
3     const oldUrl = (await target.page()).url()
4     const newUrl = target.url()
5     console.log('Změna URL z ' + oldUrl + ' na ' + newUrl)
6   }
7 })
```

Ukázka kódu 14: Zachycení změny adresy URL tabu

Provedení změny adresy URL jsme už viděli v uk. k. 10. Pro připomenutí: stačí zavolat členskou metodu `goto` objektu `page` a jako jediný parametr předat adresu URL.

Změna aktivního tabu

Pro zachycení změny aktivního tabu bohužel neexistuje žádné API v Puppeteeru. Dokonce ani není možné zjistit, který tab je momentálně aktivní. Puppeteer alespoň umožňuje spustit vlastní JavaScript uvnitř stránky s vrácením výsledku. Toho můžeme využít a na všech otevřených stránkách zavolat `document.visibilityState === 'visible'`, aktivní stránka by nám měla vrátit `true`. Problémem stále zůstává, jak se o změně aktivního tabu dozvědět v okamžiku změny. To s pouhým využitím Puppeteeru nejde a musíme se spokojit s následujícím:

- Po otevření nového tabu je nově otevřený tab aktivním tabem.
- Při zachycení události ohlašující změnu adresy URL je nezbytné ověřit, zda se nezměnil aktivní tab.
- Při zachycení události ze skupiny „ViewportEvents“ je nutné zkontrolovat, zda před ní nedošlo ke změně aktivního tabu.

Dodržením předchozích bodů se spolu se zachycenou událostí dozvíme o provedené změně aktivního tabu.

A.2 obsahuje kód, který nám vrátí momentálně aktivní tab, myšlenka kódu byla inspirována diskuzí na GitHubu [58].

Situace je zcela odlišná, pokud chceme nastavit určitý tab jako aktivní. Pro tento úkon existuje v API Puppeteeru metoda.

```
1 await page.bringToFront()
```

Ukázka kódu 15: Nastavení tabu jako aktivního

3.2.2 ViewportEvents

Jak již víme, API Puppeteeru nám umožňuje vykonat JavaScript uvnitř Chromu. Navíc, a to je pro zachycení událostí z viewportu nepostradatelné, můžeme „zveřejnit“ libovolnou metodu našeho kódu do `window` objektu Chromu. To znamená, že uvnitř okna prohlížeče můžeme vykonat kód, který zavolá zpátky metodu (i včetně předání parametrů) v našem kódu pracujícím s Puppeteerem.

Případné zájemce o kód odkážeme do appendixu – A.1 obsahuje společnou kostru pro zachycení všech akcí. Součástí A.1 je i popis částí kostry. Upozorňujeme však, že kód postrádá zejména algoritmy pro získání identifikátorů elementů jakými jsou selektory nebo lokátory.

Přesuneme se ke způsobům přehrávání získaných akcí, jak množné číslo naznačuje, způsoby se budou lišit, a to nejen podle druhu akce. API Puppeteeru umí pracovat pouze se selektory, pokud chceme pracovat s lokátory, musíme si vytvořit vlastní metody, které je převádějí na XPath. Neexistence jednotného API pro různě identifikované elementy vyžaduje neustále dodávat vlastní rozšíření do standardní verze Puppeteeru, nebo využívat nestandardní vlastní verzi Puppeteeru opatřenou o tato rozšíření. Ani jedna z těchto variant není ideální, s využitím první varianty alespoň nepřijedeme o kompatibilitu, z tohoto důvodu se domníváme, že první varianta je lepší volbou.

Nyní ukážeme konkrétní způsoby přehrávání akcí, pro jednoduchost se omezíme pouze na identifikaci pomocí selektorů. Metody převádějící vybrané lokátory na XPath s návratovou hodnotou odpovídajících elementů jsou k nahlédnutí v A.4.

Kliknutí

API Puppeteeru obsahuje metodu pro provedení kliknutí.

```
1 await page.click(selector)
```

Ukázka kódu 16: Kliknutí na element odpovídající selektoru

Přejetí myši

Situace je zde obdobná jako pro kliknutí.

```
1 await page.hover(selector)
```

Ukázka kódu 17: Přejetí myši na element odpovídající selektoru

Odeslání formuláře

Pro odeslání formuláře neexistuje API Puppeteeru. Víme, že je možné nechat vykonat JavaScript uvnitř prohlížeče, což se nám hodí, protože Web API disponuje jednoduchou metodou `submit` pro odeslání formuláře [34].

```
1 await page.$eval(selector, form => form.submit())
```

Ukázka kódu 18: Odeslání formuláře pomocí Web API

Dle API Puppeteeru, v uk. k. 18 odpovídá proměnná `form` návratové hodnotě `document.querySelector(selector)`.

Rolování

Nápodobně jako u odesílání formuláře i rolování je možné provést pomocí Web API. `window` objekt pro rolování stránky zahrnuje hned několik funkcí k tomuto účelu: `scrollTo`, `scrollBy`, `scrollByLines` a `scrollByPages` [78]. První funkce z tohoto seznamu roluje stránku absolutně vůči levému hornímu rohu, ostatní fungují relativně vůči aktuálnímu stavu.

```
1 await page.evaluate(() => window.scrollTo(x, y))
```

Ukázka kódu 19: Rolování stránky

Pokud bychom potřebovali rolovat nikoliv celou stránku ale element, máme k dispozici funkce `scrollTo` a `scrollBy` [19], které fungují stejně jako varianty pro `window` objekt.

Dvojitě kliknutí

Zde opět nemáme podporu API Puppeteeru, a tak musíme využít Web API. Připomeňme si nejprve obvyčejné kliknutí, to je možné provést voláním členské funkce `click` objektu `HTMLElement` [33]. Takto se kliknutí provede, i když využije jeho podporu v API Puppeteeru.

Pro dvojitě kliknutí bychom očekávali existenci metody `dblclick`, ať už ve Web API, nebo v API Puppeteeru. Taková metoda neexistuje a proto musíme:

1. Vytvořit událost reprezentující dvojitě kliknutí.
2. Získat element, na který událost aplikujeme.
3. Spustit vytvořenou událost se získaným elementem.

```
1 await page.evaluate(() => {  
2   let evt = new Event('dblclick')  
3   let el = document.querySelector(selector)  
4   el.dispatchEvent(evt)  
5 })
```

Ukázka kódu 20: Dvojitě kliknutí na element odpovídající selektoru

Označení textu

Označením textu rozumíme událost „select“, která je dostupná pouze pro elementy `<input type="text">` a `<textarea>`. Touto událostí se zabýváme, protože chceme získat možnost zaznamenat a opětovně simulovat výběr textu, který se může měnit. Navíc po výběru textu často následuje událost kopírování (zápis textu do schránky). Kopírovat pak můžeme aktuální výběr místo staticky uloženého textu.

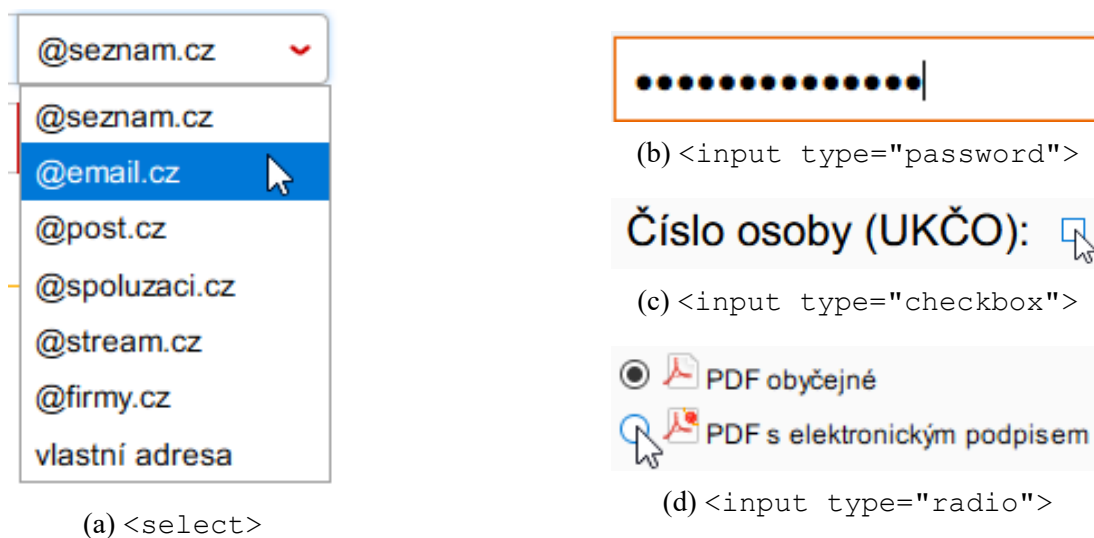
```
1 await page0.$eval(selector, el => {  
2   el.setSelectionRange(event.selectionStart,  
3                           event.selectionEnd,  
4                           event.selectionDirection)  
5 })
```

Ukázka kódu 21: Výběr textu

V uk. k. 21 předpokládáme, že proměnná `event` obsahuje informace o události „select“. Konkrétně `selectionStart` i `selectionEnd` jsou indexy textu. V `selectionStart` výběr textu započal, znak na indexu `selectionEnd` do výběru už nepatří. `selectionDirection` udává směr, kterým byl výběr proveden.

Událost „change“

Problémem této události je, že zahrnuje změny hodnot v elementech různého typu.



Obrázek 13: Některé elementy, jejichž změna hodnoty vyvolá událost „change“

Puppeteer nám dává k dispozici metodu `select`, která umí vybrat hodnotu v případě (a). Pro přehrání vytisknutí hesla v (b) existuje metoda `type`, které můžeme nastavit i volitelné zpoždění mezi jednotlivými údery.

```
1 page.select(selector, ...values)
```

(a)

```
1 page.type(selector, text, {delay: durationMs})
```

(b)

Ukázka kódu 22: Přehrávání události „change“

Pro poslední dva elementy obr. 13(c) a 13(d) lze použít kliknutí, které je možné provést nám již známou metodou `click`.

Zápis/čtení textu do/ze schránky (události „copy“ a „paste“)

Předtím, než se začneme zabývat kopírováním a vkládáním, bychom rádi uvedli čtenáře do aktuálního stavu udělování oprávnění pomocí CDP, ať už nepřímo použitím Puppeteeru, nebo nativně s pouhým využitím CDP.

Dle Puppeteer API by mělo být možné udělit oprávnění „push“, nicméně při pokusu o získání tohoto oprávnění je program ukončen s výjimkou „Error: Unknown permission: push“. Pro ověření tohoto tvrzení je možné spustit kód v A.3.

S oprávněními Puppeteeru „clipboard-read“ a „clipboard-write“ pro práci se schránkou je situace podobná jako u oprávnění „push“. Zřejmě bychom očekávali, že bez oprávnění „clipboard-read“ („clipboard-write“) není možné číst (měnit) obsah schránky. Tak tomu ale není.

Experimentálním ověřením s nahlédnutím do zdrojového kódu Puppeteeru a online dokumentace CDP [14] jsme zjistili, že:

- Při neudělení žádného oprávnění je možné zapisovat do schránky, při pokusu o čtení se zobrazí výzva pro udělení/neudělení oprávnění pro čtení. Připravený kód, kterým je možné toto ověřit se nachází v A.6.

- Puppeteer nabízí oprávnění „clipboard-read“ a „clipboard-write“, CDP definuje oprávnění „clipboardReadWrite“ a „clipboardSanitizedWrite“. Při pokusu získání oprávnění „clipboard-read“ nebo „clipboard-write“ Puppeteer vždy zavolá CDP funkci `Browser.grantPermissions` s parametrem `permissions` obsahujícím pouze „clipboardReadWrite“.

Volání této funkce, ať už přímo nativně, nebo nepřímo s použitím Puppeteeru, má za efekt to, že při pokusu zápisu do schránky se program ukončí s výjimkou „Error: Evaluation failed: DOMException: Write permission denied.“, více viz A.7. Čtení naopak funguje bezproblémů viz A.8.

- API Puppeteeru, resp. Puppeteer nijak nevyužívá oprávnění „clipboardSanitizedWrite“ definované CDP, avšak udělením tohoto oprávnění získáme práva k zápisu. Konečná funkční ukázka se nachází v A.5.

Následující dvě tabulky shrnují chování.

Oprávnění CDP	Odpovídající oprávnění Puppeteeru
clipboardReadWrite	clipboard-read clipboard-write
clipboardSanitizedWrite	

Tabulka 8: Oprávnění pro schránku definované CDP a Puppeteerem

Udělená oprávnění	Čtení	Zápis
bez oprávnění	✗	✓
clipboardReadWrite	✓	✗
clipboardSanitizedWrite	✗	✓
clipboardReadWrite a clipboardSanitizedWrite	✓	✓

Tabulka 9: Skutečný význam oprávnění CDP pro schránku

Důvodem těchto problémů může být, že online dokumentace CDP [14] uvádí celé API pro oprávnění jako experimentální.

4 Řešení

V této kapitole se detailně zaměříme na vlastní řešení navržené s využitím Puppeteeru. Začneme s funkčními a nefunkčními požadavky kladenými na naše řešení, od těch se přesuneme k designu: popisu částí řešení a způsobu komunikace mezi nimi, včetně diskuze o vybraných technologiích. Poté se podíváme na funkčnost a porovnáme ji s konkurenčními produkty. Dále se v této kapitole, konkrétně v části 4.8, nachází dokumentace pro uživatele. Pro zájemce o detailnější popis implementace je připravena podsekce 4.9.

4.1 Požadavky

V této části uvedeme specifikaci funkčních a nefunkčních požadavků pro naše řešení.

4.1.1 Funkční

(F1) Nahrávání (zaznamenání) akcí

Řešení musí podporovat nahrávání běžně prováděných úkonů v prohlížeči. Za tyto úkony považujeme: kliknutí, přejetí myši přes element, vyplnění formuláře, odeslání formuláře, rolování a načtení nové stránky.

Mezi další úkony, které by mohly být podporovány patří: práce se schránkou – kopírování a vkládání, otevření nového tabu a zavření existujícího tabu.

(F2) Přehrávání (opětovné vykonání) akcí

Všechny akce, které je možné nahrát, musí být možné i přehrát.

(F3) Generování kódu

Řešení musí umožnit, mimo okamžitého přehrávání i vygenerovat skript kompatibilní s Puppeteerem, který je funkčně ekvivalentní s přehráváním.

(F4) Použitelná identifikace elementů

Řešení musí podporovat nalezení dalších identifikátorů elementu mimo běžně používané (jednoznačná kombinace CSS tříd, argument id, absolutní popis cesty pomocí XPath, ...).

(F5) **Lokální i vzdálené připojení k prohlížeči**

Kromě standardního vytvoření nové lokální instance prohlížeče, ve které jsou akce nahrávány nebo přehrávány, musí být možné se připojit i na existující instanci spuštěnou na vzdáleném počítači.

(F6) **Nastavení**

Nahrávání a přehrávání akcí bude konfigurovatelné, k dispozici musí být alespoň nastavení:

- výběru typu akcí k nahrávání
- výběru akcí pro přehrávání
- zpoždění akcí při přehrávání
- specifik pro generování kódu (přidat řádky importující knihovny, ...)

4.1.2 Nefunkční

Nefunkční požadavky odpovídající funkčním požadavkům mají vedle svého názvu uvedené číslo funkčního požadavku.

(N1) **Využití Puppeteeru odp. (F1)–(F3)**

Implementace musí být provedena s použitím knihovny Puppeteer, jedná se o jádro práce, pro ostatní knihovny a technologie existuje mnoho automatizačních produktů viz 3.1 Konkurenční řešení.

(N2) **UI odp. (F1)–(F6)**

Celé řešení bude možné ovládat z uživatelského rozhraní vhodně škálovaného i na high DPI monitorech.

4.2 Design

Řešení se skládá ze dvou celků. Prvním je knihovna pro přehrávání a nahrávání akcí vytvořená s běhovým prostředím Node.js v JavaScriptu, dále označovaná jako backend. Vzhledem k implementaci Puppeteeru v TypeScriptu¹ jsme se rozhodli pro JavaScript. Druhým celkem je okenní WinForms aplikace implementovaná v C# poskytující UI, tu nazveme frontend.

¹ JavaScript se statickým typováním spravovaný firmou Microsoft [69].

C# a WinForms jsme vybrali z těchto důvodů:

1. Jednoduchost

Jak uvádí dokumentace firmy Microsoft [68], která stojí za vznikem C#, jedním z designových cílů od verze 1.0 byla snaha o „jednoduchý, moderní, univerzální objektově orientovaný jazyk“. Stejně jako C# i návrh UI ve WinForms považujeme za jednoduchý, a to zejména díky obrovskému množství vestavěných komponent (ProgressBar, PropertyGrid, ...) a vývoji pomocí drag and drop. Cílem této práce není vytvářet UI podle grafického návrhu a navíc WinForms obsahuje většinu námi potřebných komponent¹ a všechny nutné funkce.

2. Podpora vysokého DPI

Vysoká dostupnost cenově příznivých monitorů s řádově vyšším DPI než standardních 96 si žádá vytvářet škálovatelné aplikace. Tyto však nejsou pro WinForms žádný problém, škálování je možné zapnout voláním systémové metody `SetProcessDpiAwareness` [64] z konstruktoru hlavního formuláře [3].

3. Vyzrálost

Součástí první verze .NET Framework vydané v únoru 2002 byl WinForms. Dnes je stále její součástí, ač je tomu nyní 19 let. O výběru technologie pro UI by se jistě dalo diskutovat – pouze samotný Microsoft stihl vydat WPF a UWP, WPF slouží výhradně pro vývoj UI [76] a UWP je, spolu s vývojem UI, přímo univerzální platforma pro vývoj Windows aplikací [70]. Narozdíl od WinForms, WPF i UWP fungují pouze na Windows².

Na Internetu se často vyskytují otázky týkající se končícího životního cyklu WinForms [2, 38, 80–82]. Místo souhlasu s tím, že WinForms jsou mrtvé, se domníváme, že jejich stáří přispívá ke stabilitě a neměnnosti API, které může způsobit i ztrátu kompatibility mezi jednotlivými verzemi.

Backend a frontend mezi sebou potřebují vzájemně komunikovat. Např. frontend spouští nahrávání, tj. odesílá backendu pokyn ke spuštění nahrávání. Backend naopak při nahrávání ohlašuje zachycenou událost do frontendu.

¹ Chybí pouze textový editor se zvýrazňováním syntaxe, a tak využijeme externí viz 4.9.2.1, 4.9.2.4.

² WPF (Windows Presentation Foundation) vyžadují ke svému běhu .NET Framework 3.0, jeho otevřená implementace Mono [35] sice na svém webu deklaruje kompatibilitu s verzí 4.7, explicitně však uvádí, že WPF nepodporuje [16].

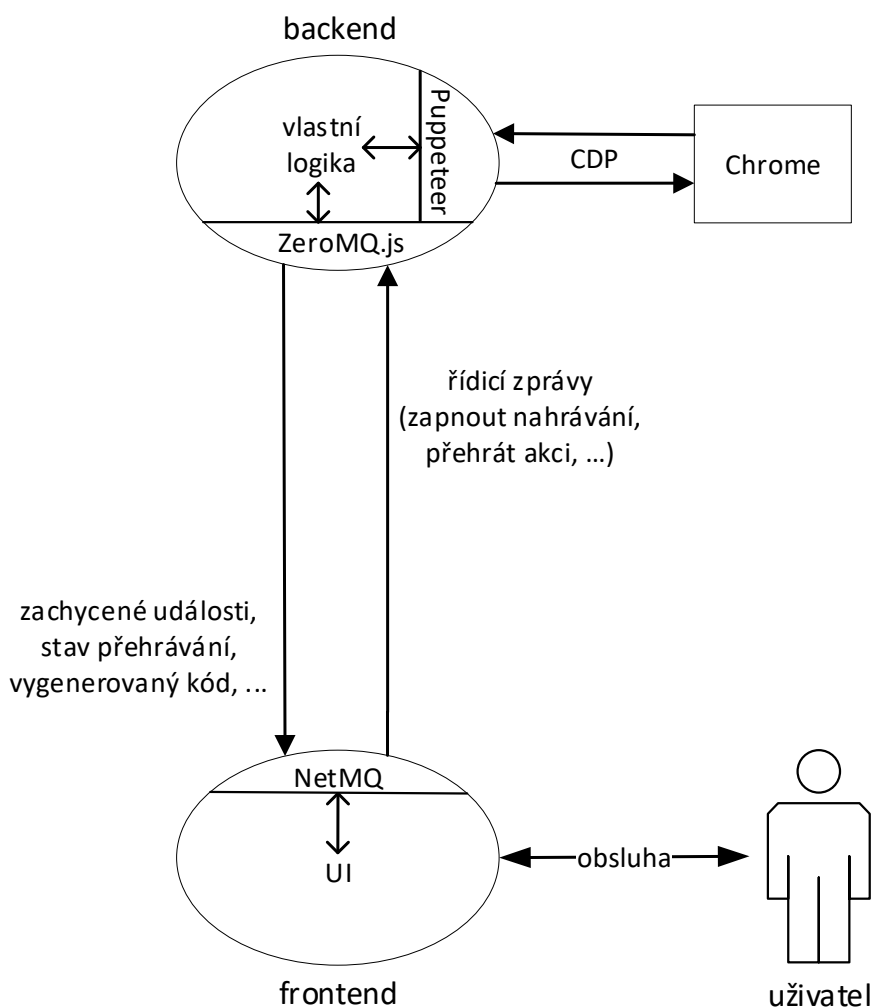
UWP (Universal Windows Platform) je z hlediska osobních počítačů dle oficiální dokumentace [11] kompatibilní pouze s Windows 10.

Pro komunikaci mezi dvěma procesy bychom mohli využít BSD sockety, které mají nativní podporu v API Node.js [36] i v API .NET Frameworku [1], pro tyto účely ale existuje řada knihoven.

Vybrali jsme knihovnu ZeroMQ [83] pro tyto její výhody:

- Existence mnoha nativních portů, případně language bindingů pro spoustu jazyků [85]. Pro nás zásadní je existence portu NetMQ [29] pro C#¹ a bindingu ZeroMQ.js [30] pro JavaScript.
- ZeroMQ nevyžaduje message broker, vrstvu mezi frontendem a backendem zpracovávající zprávy [74] viz A.9.
- Jednoduché API umožňující přímé odeslání zpráv jako textových řetězců viz A.9.

Design řešení včetně jeho částí a komunikace mezi frontendem a backendem ilustruje následující diagram.



Obrázek 14: Diagram řešení

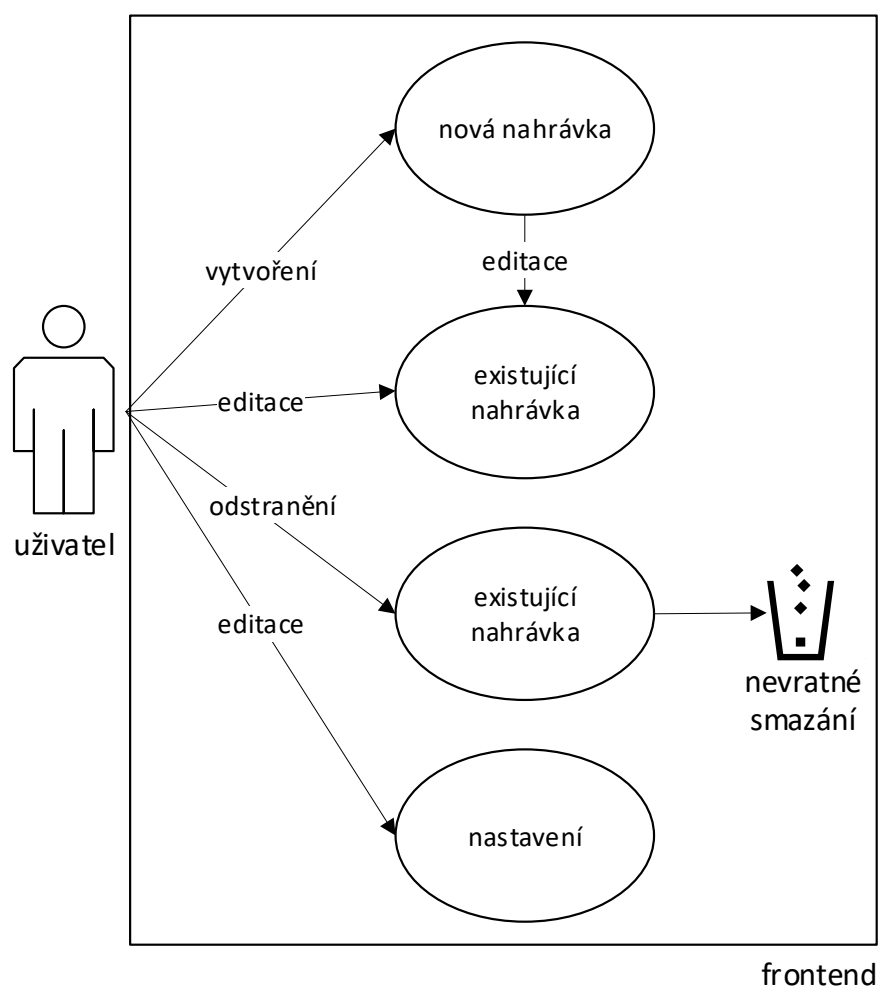
¹ Existuje i binding clrzmq4 [28], dokumentace [84] ale doporučuje použít NetMQ.

4.3 Scénáře použití

Tato část obsahuje několik diagramů, které ilustrují možnosti uživatelů pro využití řešení.

Hlavní okno

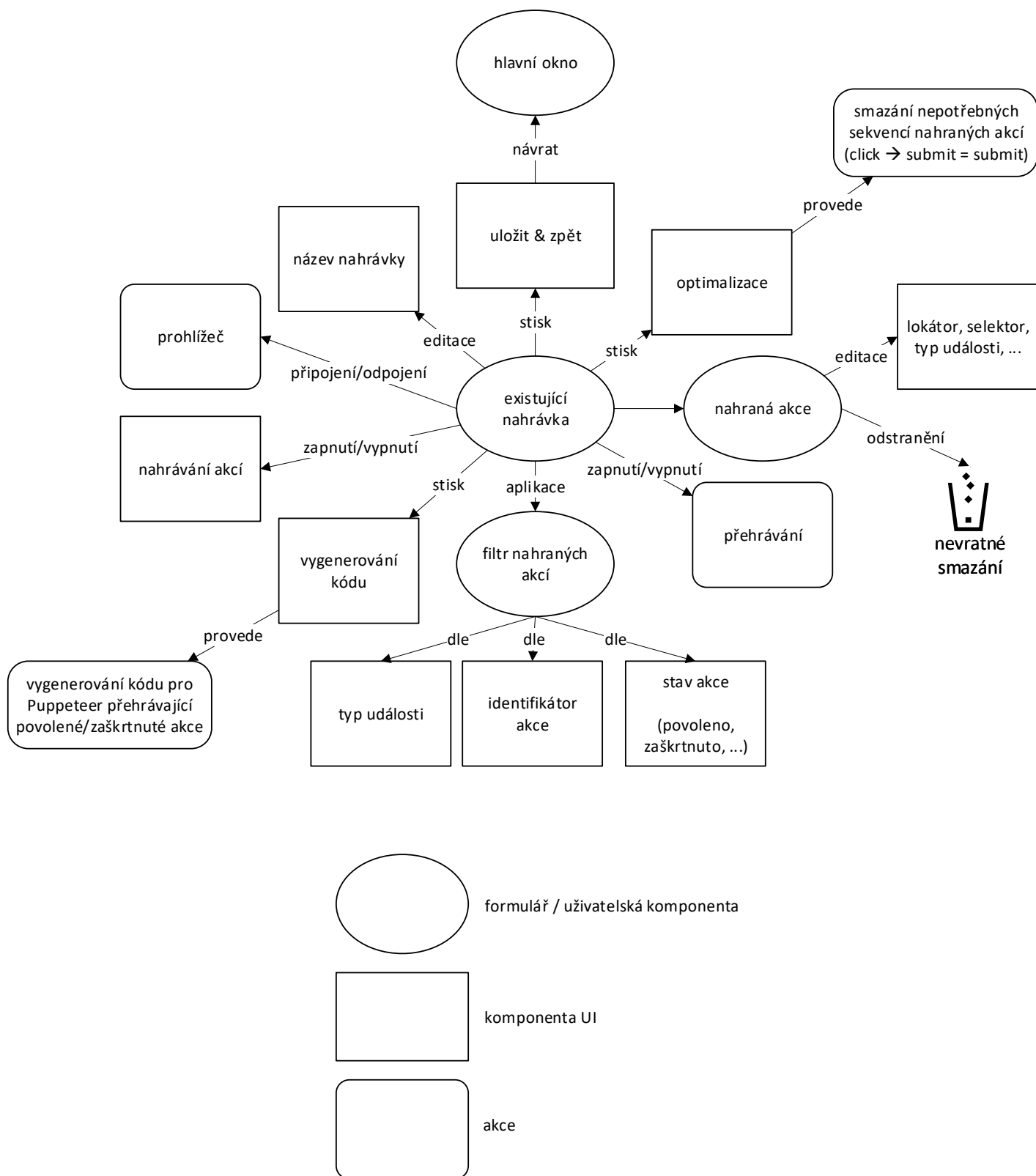
Hlavní okno reprezentuje stav ihned po spuštění.



Obrázek 15: Diagram hlavního okna

Editace nahrávky

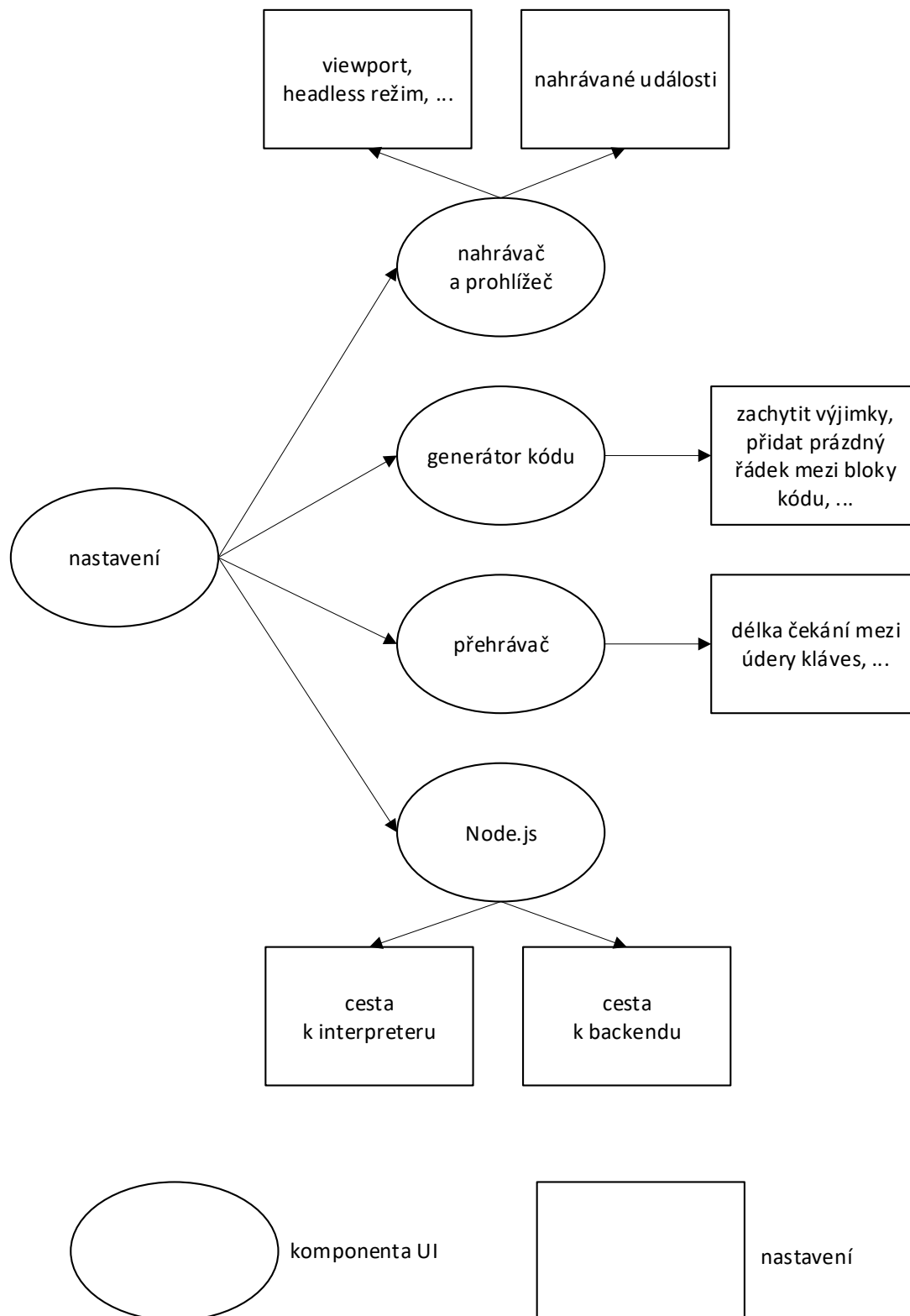
Zde rozvedeme konkrétní možnosti pro editaci existující nahrávky.



Obrázek 16: Diagram editace existující nahrávky

Editace nastavení

Nyní ukážeme některé konkrétní možnosti editace nastavení.



Obrázek 17: Diagram editace nastavení

4.4 Porovnání funkcionality oproti konkurenčním řešením

Ještě předtím, než začneme porovnávat, bychom rádi upozornili na zásadní odlišující faktor vlastního řešení, a sice využití Puppeteeru. Ostatní řešení zmíněná v této práci využívala Selenium, případně byla proprietární. (Výjimkou je Headless Recorder, ten je postavený nad Puppeteerem, ale neumí přehrávání – nepovažujeme ho za plnohodnotné řešení.)

Události viewportu

Událost	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
click	✓	✓	✓	✓	✓
dblclick	✓	✓	✓	vygeneruje kód pro 2x click	✓
input	✓	✓	✓	✓	✓
change	✓	✓	✓	✓	✓
select	✗	✗	provádí relativní pohyb kurzoru simulující výběr textu	✗	✓
submit	✓	✓	provede akci spouštějící submit	provede akci spouštějící submit	✓
scroll	✓	✗	✓	✗	✓
copy, paste	pracuje s vlastní interní schránkou	pracuje s vlastní interní schránkou	✓	✗	✓
mouseover	pomocí kontextové nabídky	✗	simulace pohybu kurzoru	✗	podržením klávesy „h“

Tabulka 10: Porovnání podporovaných událostí viewportu napříč řešeními

Události okna prohlížeče

Událost	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
otevření nového tabu	implicitně s akcí elementu	implicitně s akcí elementu	✓	✗	✓
uzavření existujícího tabu	✓	✓	✓	✗	✗
změna URL adresy	ručním přidáním akce open	ručním přidáním akce open	✓	✗	✓
změna aktivního tabu	✓	✓	✓	✗	detekováno s následující událostí

Tabulka 11: Porovnání podporovaných událostí okna prohlížeče napříč řešeními

Identifikátory elementů

Typ identifikátoru	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
selektory	✓	✗	využívá proprietárně rozšířený xpath identifikující i mimo viewport	✓	✓
lokátory	✓	pouze id, name, xpath		✗	✓

Tabulka 12: Porovnání podporovaných identifikátorů elementů napříč řešeními

Čekání na

Čekání na	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
existenci elementu	✓	✓	✓	✓	✓
hodnotu atributu elementu	✗	✓	✓	✗	✗
možnost editace elementu	✓	✓	✓	✗	✗
dokončení navigace	implicitně čeká na zavolání události „load“	implicitně čeká na zavolání události „load“	✓	implicitně čeká na zavolání události „load“	✓
viditelnost elementu	✓	✓	✓	✗	✗

Tabulka 13: Porovnání podporovaných druhů čekání napříč řešeními

Možnosti exportu

Technologie	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
C# (MSTest)	✗	✓	✗	✗	✗
C# (NUnit)	✓	✓	✗	✗	✗
C# (xUnit)	✓	✗	✗	✗	✗
Java (JUnit)	✓	✓	✗	✗	✗
JavaScript (Mocha)	✓	✗	✗	✗	✗
JavaScript (New Relic Synthetics)	✗	✓	✗	✗	✗
JavaScript (Playwright)	✗	✗	✗	✓	✗

JavaScript (Puppeteer)	✗	✗	✗	✓	✓
JavaScript (WebDriver.io)	✗	✓	✗	✗	✗
JSON	✗	✓	✗	✗	✗
Katalon Studio	✗	✓	✗	✗	✗
mezikód pro výrobu nových formátů	✗	✓	✗	✗	✗
Python (App- Dynamics)	✗	✓	✗	✗	✗
Python (pytest)	✓	✗	✗	✗	✗
Python 2 (unittest)	✗	✓	✗	✗	✗
Robot Framework	✗	✓	✗	✗	✗
Ruby RSpec	✓	✓	✗	✗	✗
samostatný spustitelný soubor	✗	✗	✓	✗	✗
TypeScript (Protractor)	✗	✓	✗	✗	✗
XML	✗	✓	✗	✗	✗

Tabulka 14: Porovnání podpory generování kódu napříč řešeními

Další pokročilé funkce

Funkce	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
větvení (podmínky)	if, else if, else	if, else if, else	možnost propojení s C#	✗	✗
cykly	do, times, while, foreach	while	možnost propojení s C#	✗	✗
zvýraznění identifikovaného elementu uvnitř viewportu	✓	✓	dokonce i streamování aktuální podoby elementu	✗	✗
změna elementu existující akce klikem uvnitř viewportu	✓	✓	pomocí sofistikované součásti Ranorex Spy	✗	ruční změna, nebo nahrání nové akce
ruční přidání akce	✓	✓	✓	✗	✗
nastavení velikosti viewportu	✓	✗	✓	✗	bez dynamických změn během nahrávání
volitelné zpomalení přehrávání	✓	✓	✓	✗	✓
připojení na vzdálený prohlížeč	✗	✗	pouze přehrávání přes RDP	✗	✓

Tabulka 15: Porovnání dalších pokročilých funkcí napříč řešeními

4.5 Vývoj

Mimo software, uvedený v tab. 1, bylo k vývoji použito několik nástrojů.

Visual Studio 2019 16.0.17

Celá frontendová část je projektem Visual Studio 2019, ve kterém taktéž probíhal její vývoj. Vzhledem k tomu, že využíváme WinForms, bylo Visual Studio jasnou volbou díky existenci designru.

WebStorm 2020.2.2

Pro vývoj backendové části jsme použili WebStorm. Hlavním důvodem výběru byla podpora refactoringu a debuggeru.

npm 7.5.1

K práci s balíčky pro Node.js jsme použili výchozího správce npm. Frontendová část obsahuje několik málo jednoduchých závislostí, které nejsou pro npm problémem, navíc pro pokročilé funkce (workspaces), kterými disponuje konkurenční yarn, nemáme využití.

4.6 Testování

Již od začátku vývoje jsme řešení testovali na více počítačích s různými verzemi OS Windows, obvykle jsme distribuovali binární soubory spolu se přenositelnou verzí Chromu a Node.js pro jednoduché použití. Převážná většina testování byla provedena PC s nainstalovaným OS Windows 10 20H2 se starším hardwarem:

- CPU: AMD Athlon 64 X2 5600+
- RAM: 6GB
- GPU: NVIDIA Geforce 8800 GTS

Pro úplnost přikládáme tabulku testovaných/netestovaných verzí Windows s nastavením DPI.

Verze OS Windows	= 96 DPI	> 96 DPI
7 SP1	✗	✗
8	✓	✗
8.1	✓	✓
10	✓	✓

Tabulka 16: Otestované verze OS Windows rozdělené podle nastavení DPI

4.6.1 Odhalené problémy

Testovali jsme všechny funkce našeho řešení, na některých počítačích jsme objevili problémy, které jsme opravili.

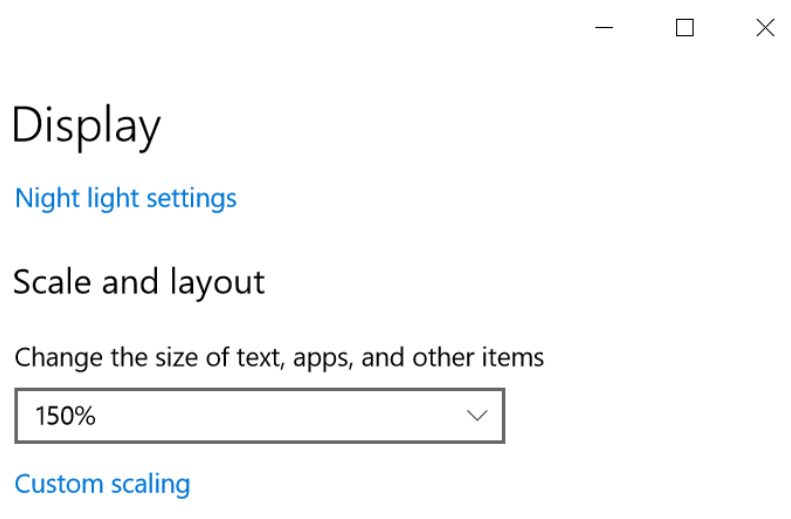
Spouštění backendu

Na jednom z testovaných počítačů s OS Windows 8.1 se standardním DPI nebylo možné stisknutím tlačítka „Connect/Start Browser“ spustit backend a prohlížeč. Zjistili jsme, že z nějakého důvodu se vůbec nespustil Node.js proces, který by měl vykonávat kód backendu. Cesty a parametry pro spuštění procesu byly správné, na jiných počítačích se problémy nevyskytovaly.

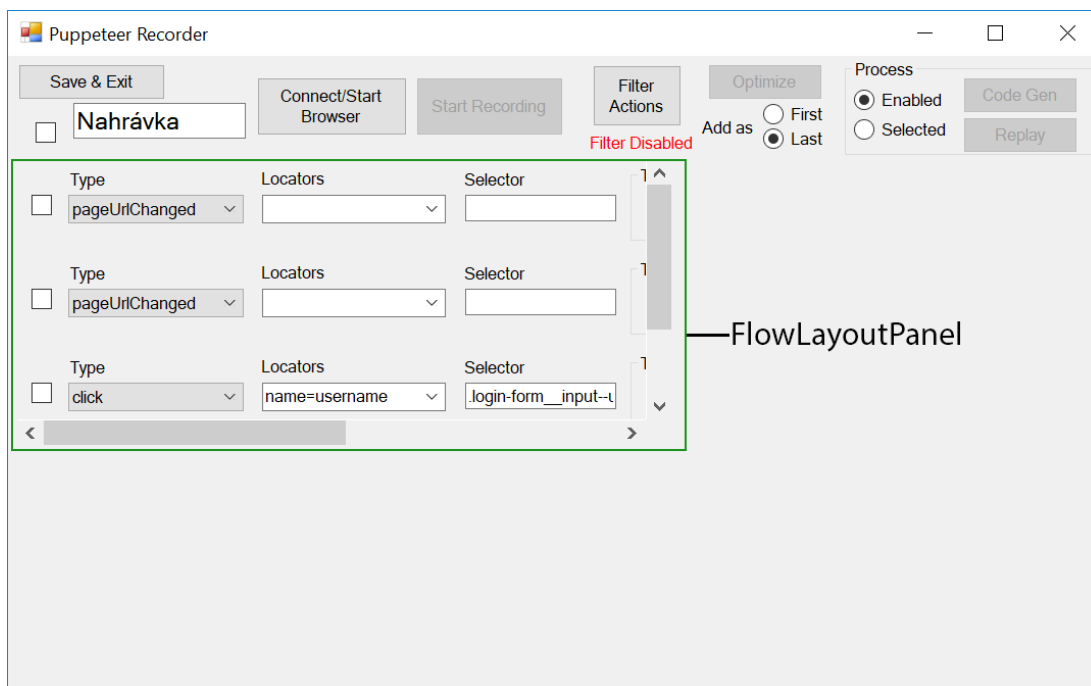
V té době jsme Node.js proces spouštěli pomocí relativní cesty k interpreteru vůči cestě spustitelného souboru frontendu. Jako spolehlivější se ukázalo použití absolutní cesty místo relativní. Použili jsme naši původní relativní cestu, kterou jsme navíc pomocí metody `Path.GetFullPath` převedli na absolutní a problém byl vyřešen.

High DPI monitor

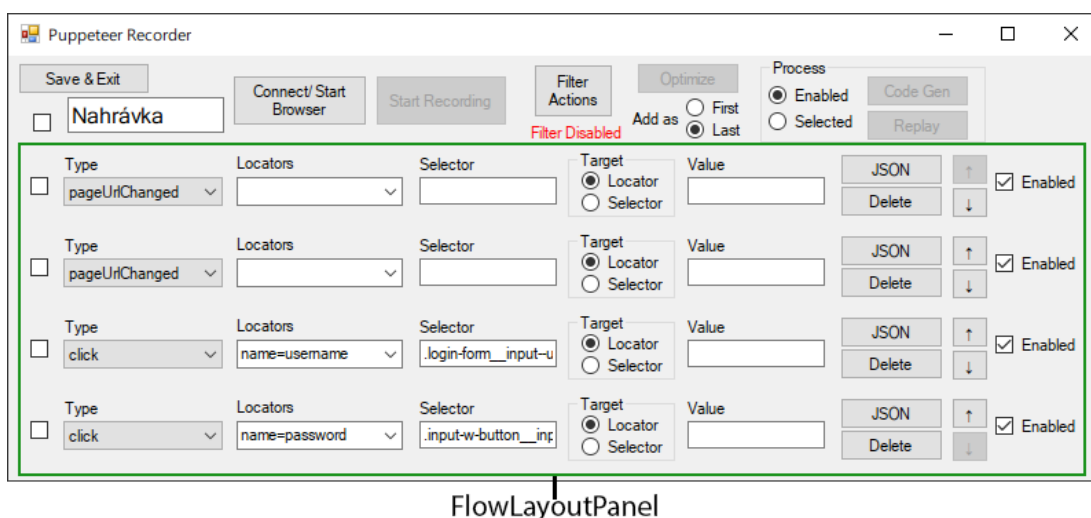
Při zapnutí škálování se na všech počítačích objevil problém s komponentou `FlowLayoutPanel`, která měla nastavenou vlastnost `Anchor` do všech rohů.



Obrázek 18: Škálování nastavené na 150%



(a) Editační UI při 150% škálování



(b) Editační UI při 100% škálování

Obrázek 19: Porovnání editačního UI při zapnutém/vypnutém škálování

Po bádání jsme přišli na to, že z neznámého důvodu se při zapnutém škálování přestane měnit velikost `FlowLayoutPanelu` spolu se změnou velikosti jeho rodiče `EditUserControl`. Problém jsme vyřešili vytvořením události `Resize` pro `EditUserControl`, kde se upravuje velikost `FlowLayoutPanelu` odpovídajícím způsobem.

4.7 Administrátorská dokumentace

Před tutoriály demonstrujícími ovládání našeho řešení definujeme systémové požadavky, závislosti a připravíme prostředí pro prvotní spuštění.

Systémové požadavky

Backend	Frontend
Node.js 15.5.0 (dle tab. 1)	.NET Framework 4.5 (dle nastavení projektu)

Tabulka 17: Systémové požadavky backendu a frontendu

Node.js od verze 14.xx.x není na Windows 7 testováno [77], z toho důvodu je zaručena kompatibilita s Windows 8 a vyšší. .NET Framework 4.5, případně jeho novější verze, je předinstalovaný na všech verzích Windows 8 a vyšších. Z těchto důvodů definujeme kompatibilitu s OS Windows 8/8.1/10.

Závislosti

Backend i frontend vyžadují ke svému běhu další knihovny dostupné z `nuget.org` a `npmjs.com`. Všechny jsou ale součástí přílohy. Pro informaci uvádíme soubory, které závislosti popisují.

Backend	Frontend
Backend\package.json	Frontend\Frontend\Frontend.csproj

Tabulka 18: Závislosti backendu a frontendu

Příprava prostředí

1. Stáhneme přílohu práce¹
2. Nainstalujeme Node.js² 15.5.0 (dle tab. 1) odpovídající architektuře OS
64bit instalátor: <https://nodejs.org/download/release/v15.5.0/node-v15.5.0-x64.msi>
32bit instalátor: <https://nodejs.org/download/release/v15.5.0/node-v15.5.0-x86.msi>

Spustitelný soubor se nachází v adresáři `Frontend\Frontend\bin\Release\Frontend.exe`

¹ Příloha je dostupná i na GitHubu [26], commit ID ... odpovídá verzi, která byla odevzdána s tímto textem.

² Mohli bychom distribuovat interpreter přímo se zdrojovým kódem, nicméně nepříjde nám jako dobrý nápad zmrazovat závislost o velikosti ~ 68 MB. Lepší varianta podle nás je, když všechny aplikace vyžadující Node.js používají jeden systémový.

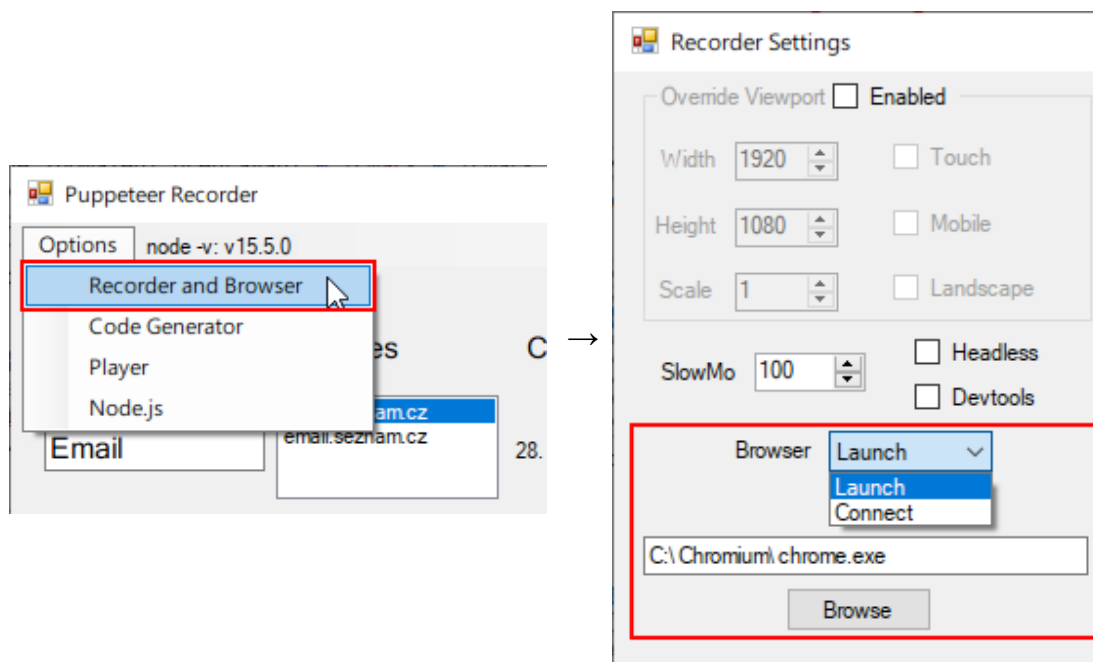
4.8 Tutoriály

Nyní se podíváme na UI frontendové části přes kterou se naše řešení ovládá. Využití uživatelského rozhraní bude demonstrováno několika tutoriály.

Tutoriály jsou vytvořené tak, aby ukazovaly základní funkcionalitu řešení, případně vysvětlovaly neintuitivní části UI. Velká část zřejmých komponent UI je pro přehlednost v dokumentaci vynechána.

4.8.1 Prvotní spuštění

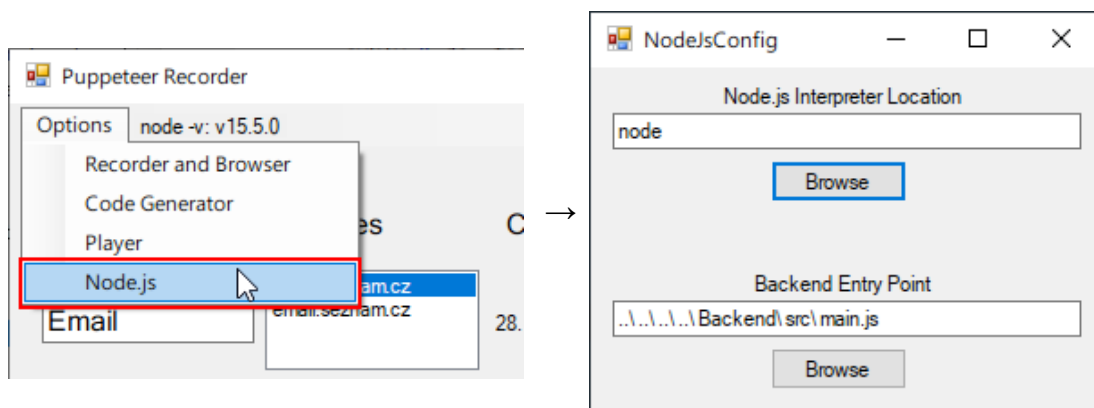
Pro správnou funkci je nutné určit typ připojení ke Chromu, nastavit cestu k interpreteru Node.js pokud není umístěna v proměnné Path proměnného prostředí. Jestliže byla změněna relativní cesta backendu vůči frontendu, tak je nutné nastavit novou cestu i k backendu. Začneme připojením Chromu.



Obrázek 20: Nastavení připojení ke Chromu

Jak je patrné z obr. 20, na výběr máme ze dvou typů připojení. „Launch“ spustí novou lokální instanci Chromu, volitelně vyžaduje zadání cesty ke spustitelnému souboru prohlížeče, její zadání je nutné pokud se spustitelný soubor Chromu nenachází ve standardním umístění. Druhý typ připojení označený jako „Connect“ se připojí k existující vzdálené instanci Chromu, více o tomto druhu připojení viz tutoriál 4.8.3 Připojení ke vzdálenému Chromu.

Poslední dvě zbývající cesty pro backend a Node.js se nastavují ve společném dialogovém okně.



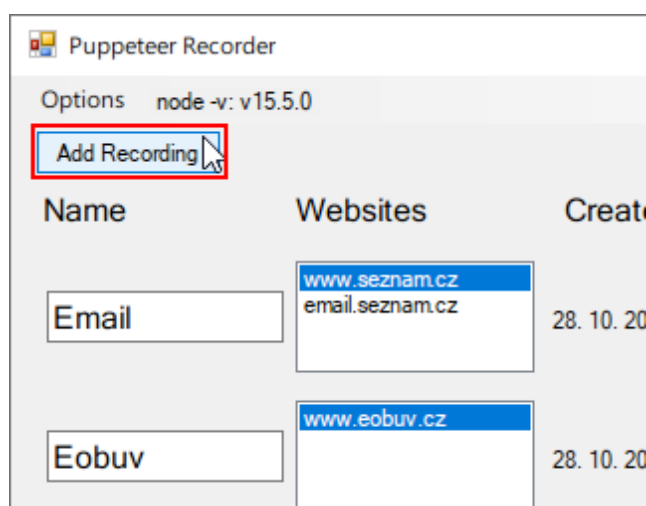
Obrázek 21: Nastavení cesty k interpreteru Node.js a backendu

Obě vyplněné cesty jsou výchozí hodnotou nastavení. Všimněme si, že jako cesta k interpreteru Node.js stačí uvést pouze „node“ pokud je jeho umístění v proměnné Path proměnného prostředí.

Cesta k backendu je popsána relativně vůči umístění spustitelného souboru frontendové části. Pokud nebyla upravena struktura projektu, cesta k backendu je správná.

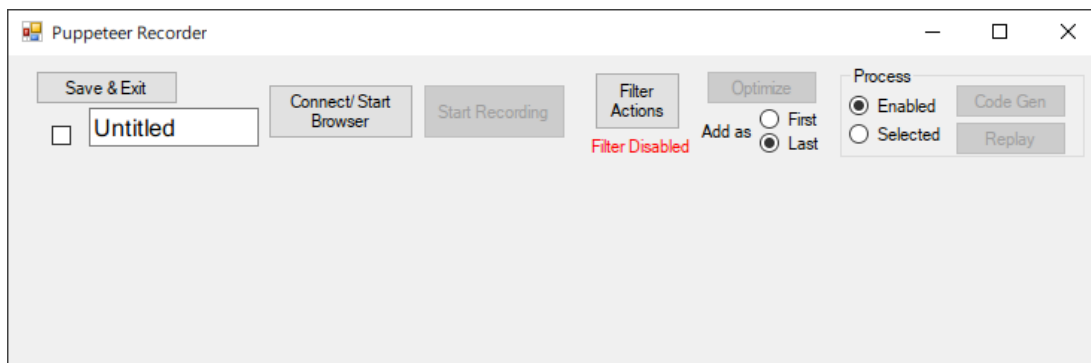
4.8.2 Nová nahrávka

Zkusíme si vytvořit jednoduchou nahrávku na které si ukážeme základní součásti editačního UI pro nahrávky.



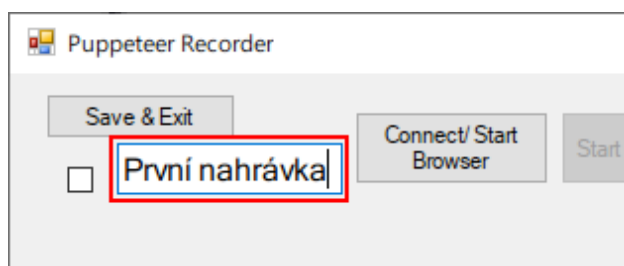
Obrázek 22: Vytvoření nové nahrávky

Po stisku zvýrazněného tlačítka na obr. 22 se změní podoba okna pro nahrávání a přehrávání akcí, okno bude vypadat takto:



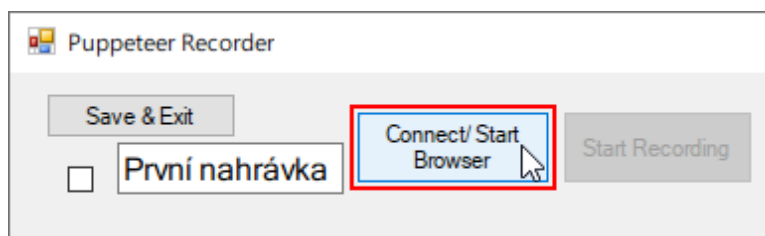
Obrázek 23: Editační UI pro nahrávky

Nyní máme prázdnou nahrávku, naším cílem bude nahrát několik akcí, přehrát je a zobrazit si jejich kód pro Puppeteer. Nejprve ale změníme jméno nahrávky z „Untitled“ na „První nahrávka“.



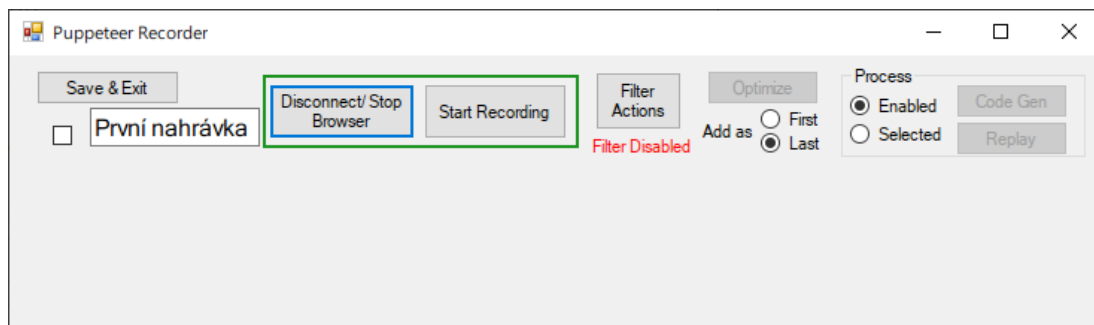
Obrázek 24: Změna jména nahrávky

Pokud si čtenář prošel tutoriál 4.8.1 Prvotní spuštění, mělo by být vše správně nastavené a Chrome v režimu spuštění lokální instance. Spustíme backend spolu s Chromem kliknutím na tlačítko „Connect/Start Browser“.



Obrázek 25: Spouštění backendu a Chromu

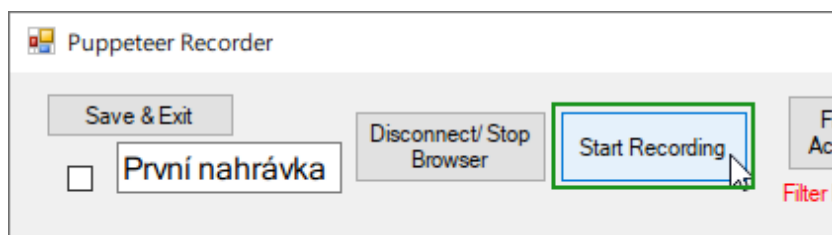
Měla by se nám otevřít nová instance Chromu s jedním prázdným otevřeným oknem. UI by mělo povolit zakázané tlačítko „Start Recording“ a vypadat takto:



Obrázek 26: Editační UI po spuštění backendu a prohlížeče

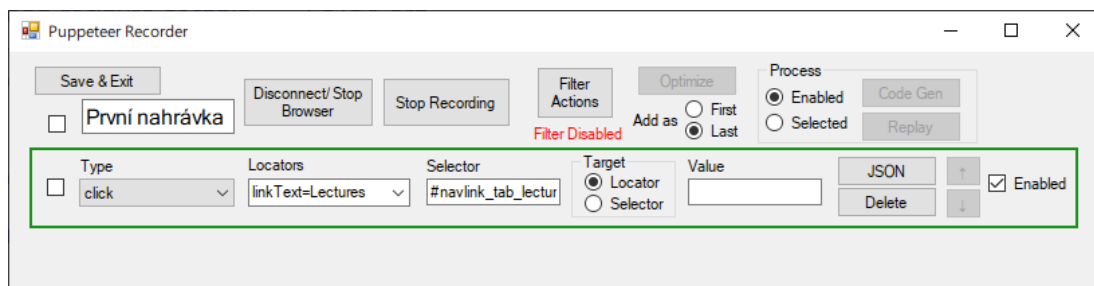
Pokud UI tak nevypadá a Chrome se neotevřel, objevila se zpráva s chybou, ta vždy popisuje konkrétní problém včetně kroků, které je nutné podniknout pro její odstranění, chyba by se ale po správném provedení kroků v tutoriálu 4.8.1 Prvotní spuštění objevit neměla.

Stiskem tlačítka „Start Recording“ spustíme nahrávání akcí provedených s otevřenou instancí Chromu. Pro seznam podporovaných akcí odkážeme čtenáře do 4.4 Porovnání funkcionality oproti konkurenčním řešením.



Obrázek 27: Zapnutí nahrávání akcí

Nahrajme si libovolný počet akcí. Nahrání akce poznáme přidáním odpovídajícího řádku do UI.



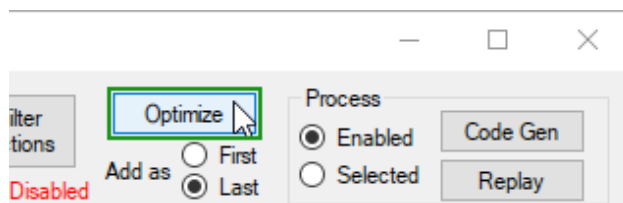
Obrázek 28: Nahrání první akce

Pro ukončení nahrávání akcí klikneme na tlačítko „Stop Recording“.



Obrázek 29: Vypnutí nahrávání akcí

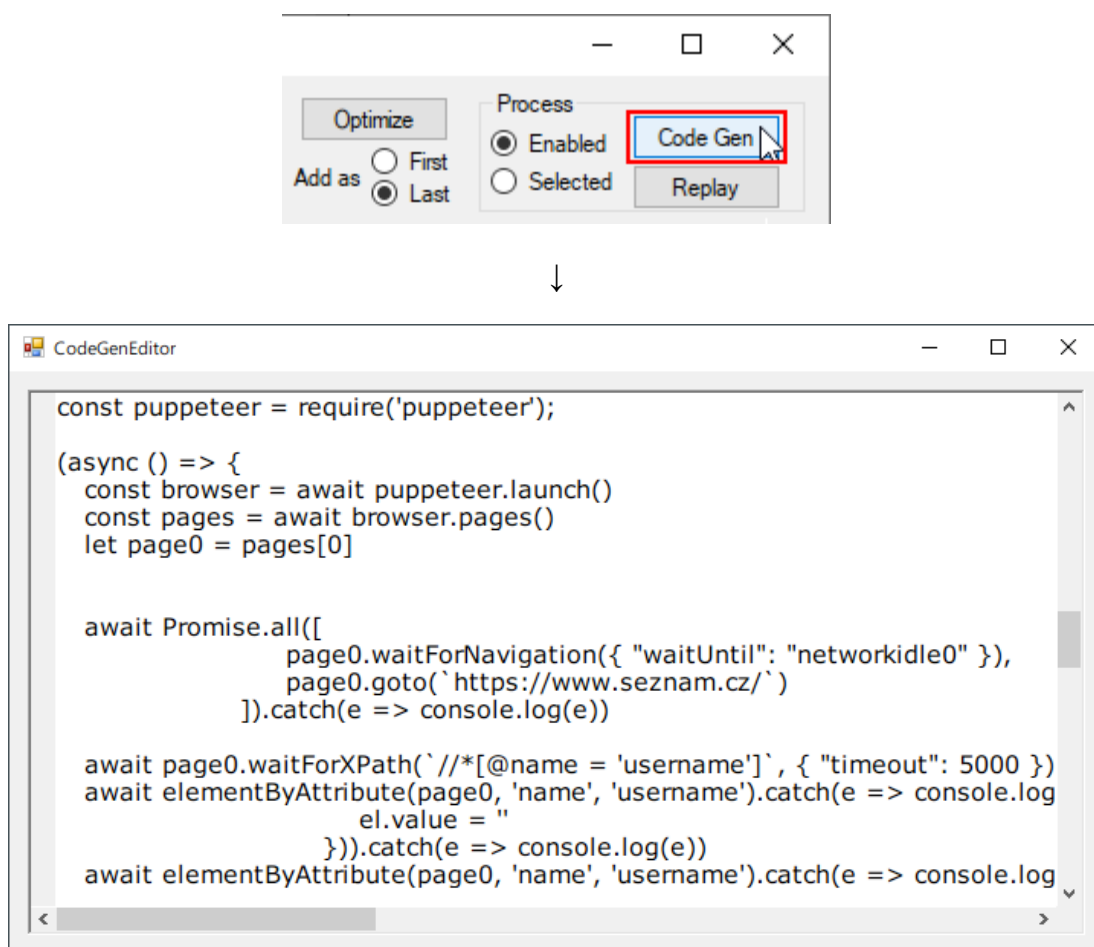
Nyní nás dělí jediný krok od generování kódu pro Puppeteer a přehrávání akcí, tím krokem je optimalizace. Při nahrávání často nahrajeme spoustu akcí z nichž značná část je pro přehrávání nepodstatná¹. Pro smazání snadno popsatelných nadbytečných akcí je zde přítomno tlačítko „Optimize“. Klikem na toto tlačítko optimalizaci nyní provedeme.



Obrázek 30: Optimalizace akcí

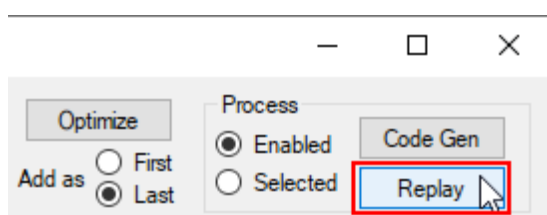
¹ Uvedeme příklad jednoduše popsatelné nepodstatné akce: Při vyplňování textového políčka se nahrají události „input“, po vyplnění a opuštění políčka se nahraje událost „change“. Pro přehrávání nám stačí pouze událost „change“. Optimalizace v tomto případě odstraní všechny události „input“. Uvědomme si, že nestačí nahrávat pouze události „change“ a úplně ignorovat události „input“. V tomto případě by se mohlo stát, že uživatel vyplní textové políčko, neopustí ho, vypne nahrávání a akce s vyplněním políčka nebude nahrána.

Aktuálně nám nic nebrání vyzkoušet si vygenerovat kód a přehrát akce.



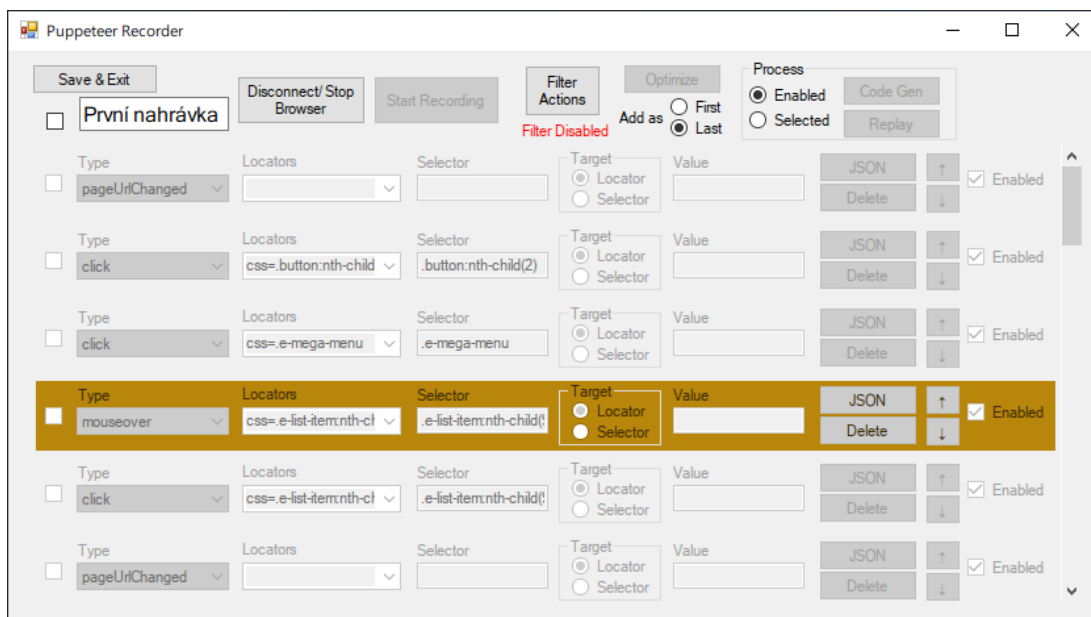
Obrázek 31: Zobrazení kódu pro Puppeteer odpovídající nahraným akcím

Akce přehrajeme kliknutím na tlačítko „Replay“.



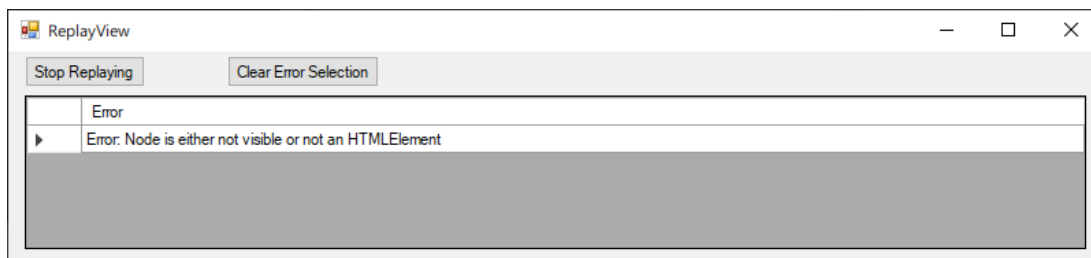
Obrázek 32: Spuštění přehrávání akcí

Po kliku se nám zvýrazní právě přehrávaná akce.



Obrázek 33: Spuštění přehrávání akcí

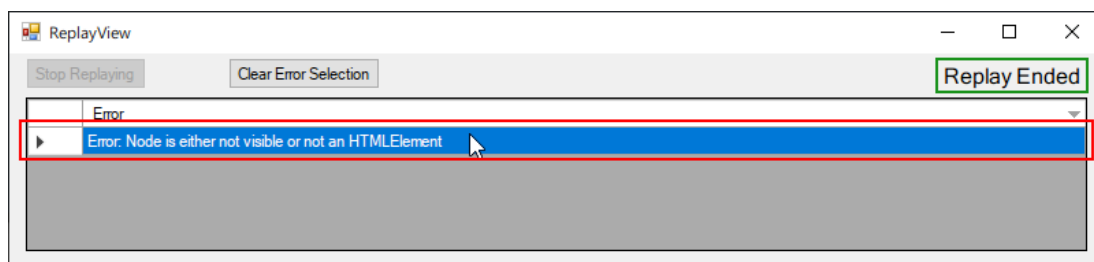
Otevře se i nové okno popisující stav probíhajícího přehrávání. Zde se zobrazí případné chyby¹ při přehrávání akcí, probíhající přehrávání je možné tlačítkem „Stop Replaying“ náhle ukončit.



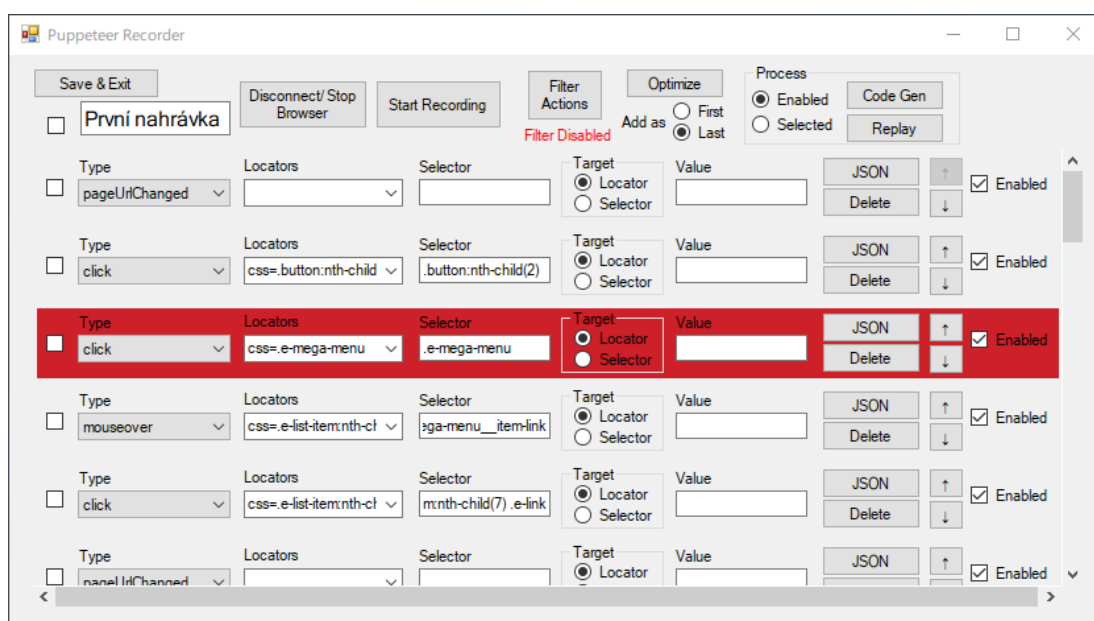
Obrázek 34: Okno popisující stav spuštěného přehrávání

¹ Chybami rozumíme neexistenci elementu a existenci více elementů pro daný identifikátor.

Při kliknutí na zprávu oznamující chybu se nám v editačním UI zvýrazní akce, která chybu vyvolala.



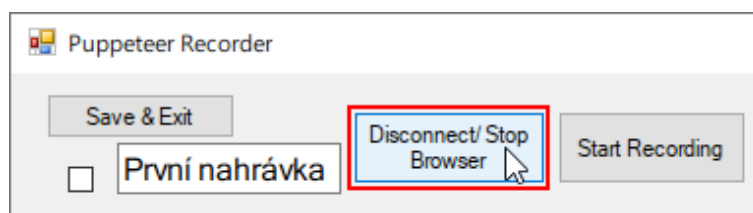
(a)



Obrázek 35: Zvýraznění akce, která vyvolala chybu

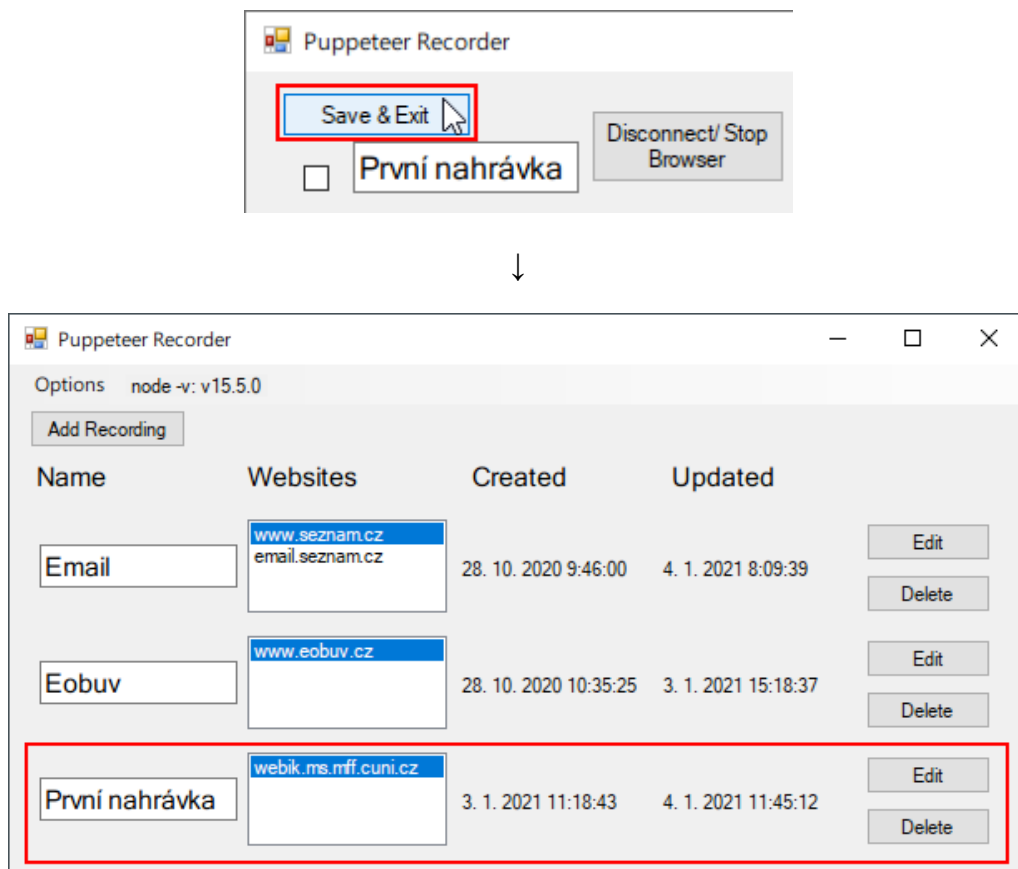
Po dokončení přehrávání se nám v okně popisujícím přehrávání objeví zpráva „Replay Ended“, ta je zvýrazněna zeleným rámečkem na obr. 35(a).

Přesuneme se od přehrávání – vraťme se nyní do hlavního spouštějícího okna frontendu. Okno popisující přehrávání zavřeme, odpojíme se od prohlížeče a vypneme backend.



Obrázek 36: Odpojení prohlížeče a vypnutí backendu

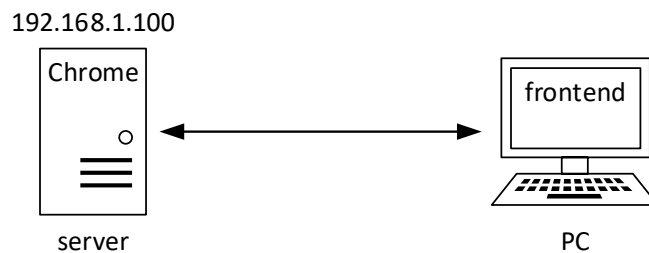
Nakonec uložíme nahrávku a vrátíme se do seznamu všech uložených nahrávek.



Obrázek 37: Uložení nahrávky a návrat do seznamu všech uložených nahrávek

4.8.3 Připojení ke vzdálenému Chromu

Pro účely tohoto tutoriálu je nutné mít připravenou jednu nahrávku a dva počítače. Předpokládáme následující schéma zapojení.



Obrázek 38: Schéma zapojení počítačů

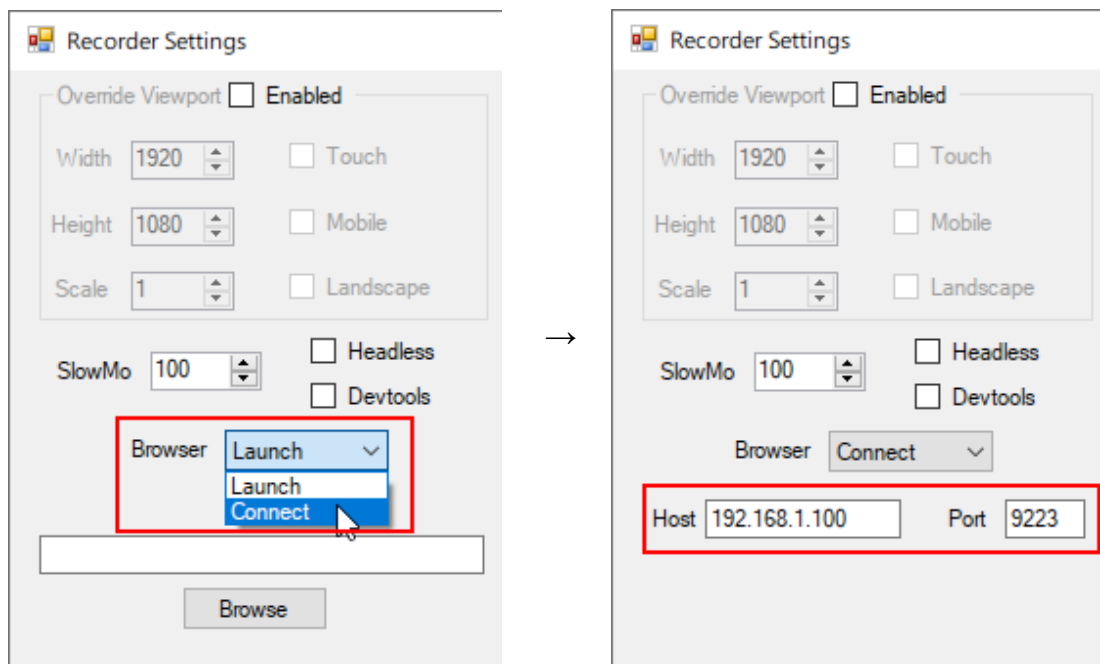
Na serveru spustíme Chrome s parametrem `--remote-debugging-port=9222`, s tímto parametrem běží ve Chromu CDP na portu 9222. Na CDP takto spuštěné instance Chromu je možné se připojit pouze z localhost [14], abychom CDP zpřístupnili

vzdáleně, musíme přesměrovat nějaký jiný vzdáleně přístupný port na port 9222. Toho je možné docílit spuštěním následujícího příkazu na serveru [32].

```
1 ssh -L 0.0.0.0:9223:localhost:9222 localhost -N
```

Ukázka kódu 23: Přesměrování vzdáleně přístupného portu 9223 na port 9222

Nyní stačí nastavit ve frontendové části připojení ke Chromu na „Connect“, zadat korektní IP adresu s portem.

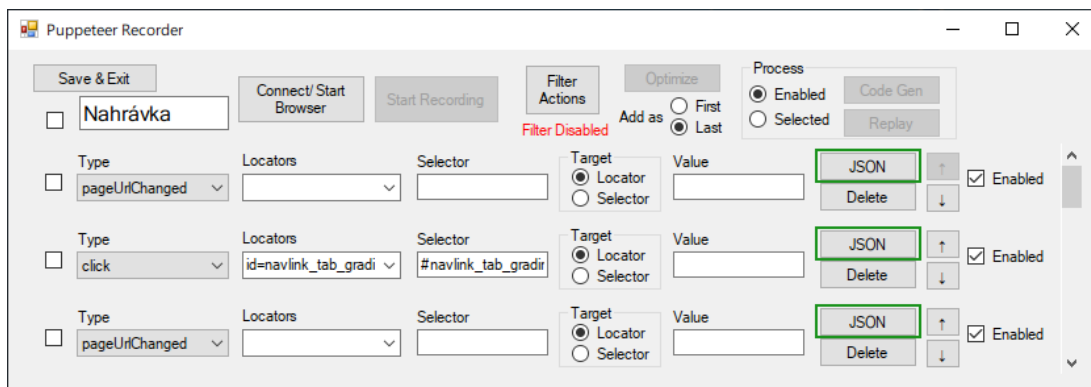


Obrázek 39: Nastavení připojení na „Connect“ s vyplněním IP adresy a portu

Můžeme si vyzkoušet přehrát připravenou nahrávku. Připojení ke Chromu a spuštění backendu tlačítkem „Connect/Start Browser“ nespustí novou instanci Chromu, ale připojí se k existující na serveru. Přehrávání nahrávky se odehraje na serveru.

4.8.4 Tlačítko JSON

Součástí každé nahrané akce je tlačítko „JSON“.



Obrázek 40: Tlačítko JSON jako součást každé akce

Po kliku na toto tlačítko se zobrazí nezpracovaná textová data ve formátu JSON popisující nahranou akci.



Obrázek 41: Data ve formátu JSON popisující nahranou akci „pageUrlChanged“

Obr. 41 popisuje podle parametru „type“ událost „pageUrlChanged“. Mezi další užitečné parametry patří „oldUrl“ a „newUrl“, jak je z jejich názvů patrné: První parametr popisuje původní adresu URL, druhý pak novou adresu, která byla načtena. Ani jeden z těchto parametrů nemá vlastní UI viz obr. 40, a proto jediná možnost nahlédnutí na konkrétní hodnoty je přes data v JSONu.

Pokud bychom potřebovali nezpracovaná data v JSONu měnit, např. chceme načíst jinou adresu URL, máme k dispozici zaškrtačací políčko s popiskem „Modifications“ (zvýrazněno zeleným rámečkem na obr. 41), které při zaškrtnutí povolí modifikace. Dovolujeme si ale upozornit, že takto provedené modifikace jsou prováděny na vlastní nebezpečí a při neplatné modifikaci je možné akci i znefunkčnit.

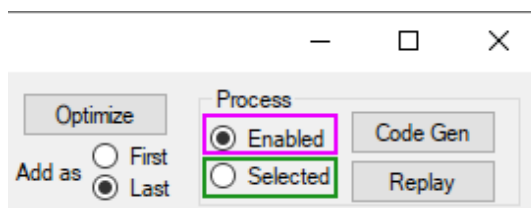


Obrázek 42: Modifikace nezpracovaných dat v JSONu

4.8.5 Výběr akcí ke zpracování

Některé nahrávky mohou obsahovat mnoho akcí, ne vždy budeme chtít všechny přehrát. Dokonce můžou nastat situace, kdy budeme chtít výhradně pro debuggovací účely přehrát jedinou akci – totéž platí nejen pro přehrávání ale i pro generování kódu. V tomto tutoriálu si ukážeme jak na to.

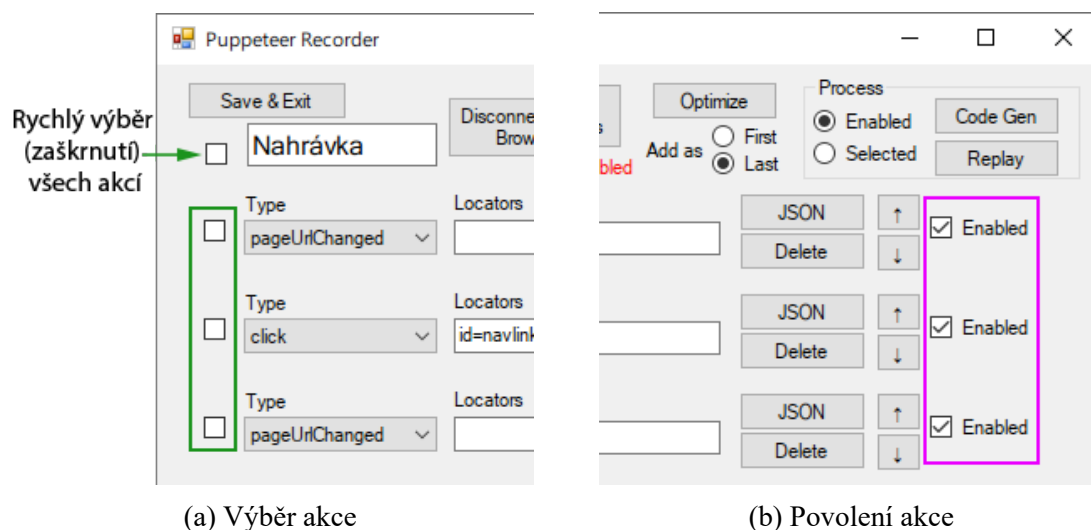
Přesuňme se do editačního UI libovolné nahrávky. Jak bylo naznačeno v úvodním odstavci, výběr akcí je možný pro přehrávání i pro generování kódu.



Obrázek 43: Přepínače pro výběr akcí ke zpracování

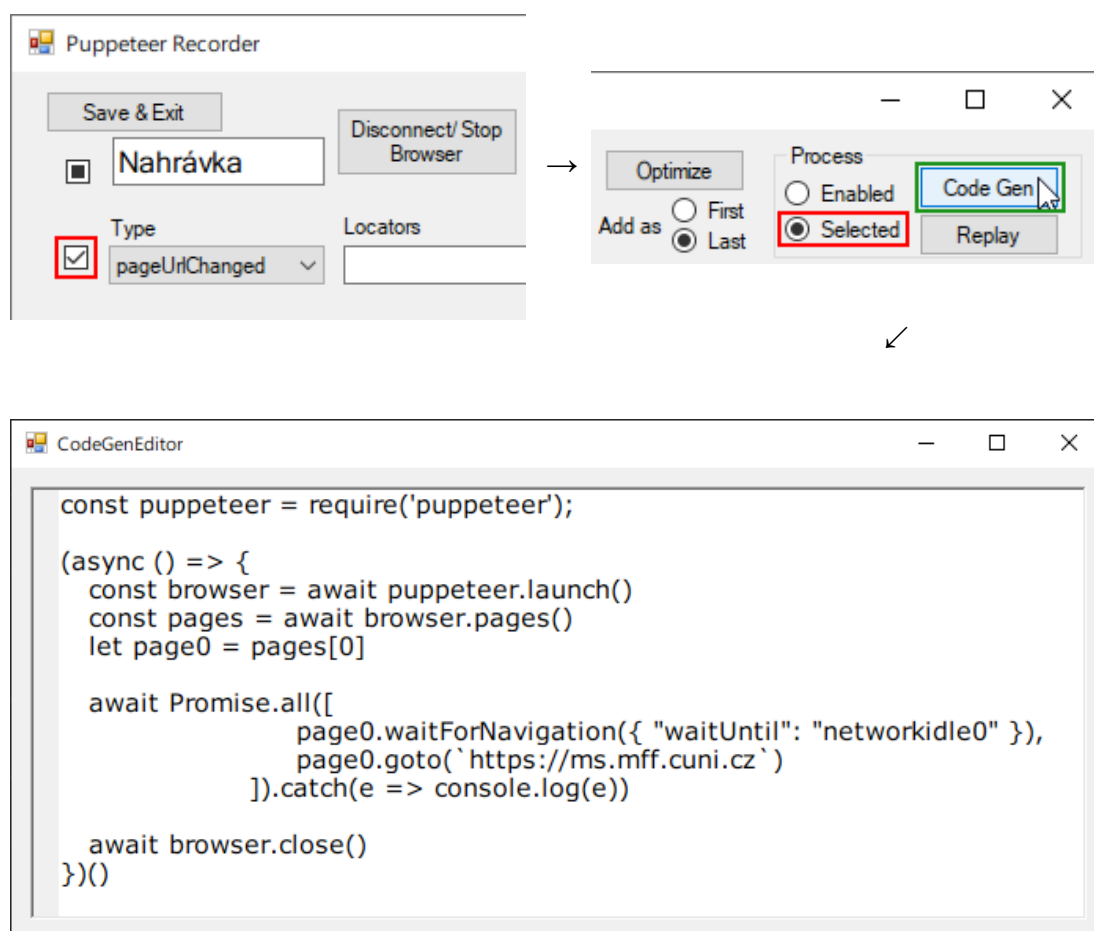
Obr. 43 nám ukazuje, že je možné zpracovat akce povolené („Enabled“), nebo akce vybrané („Selected“).

Na následujícím obrázku jsou odpovídající zaškrtnuté políčka pro výběr a povolení akcí zvýrazněna stejnou barvou jako na obr. 43.



Obrázek 44: Zaškrtnuté políčka pro výběr a povolení akcí

Na závěr tohoto tutoriálu přikládáme ukázkou vygenerování kódu jedné akce typu „pageUrlChanged“.

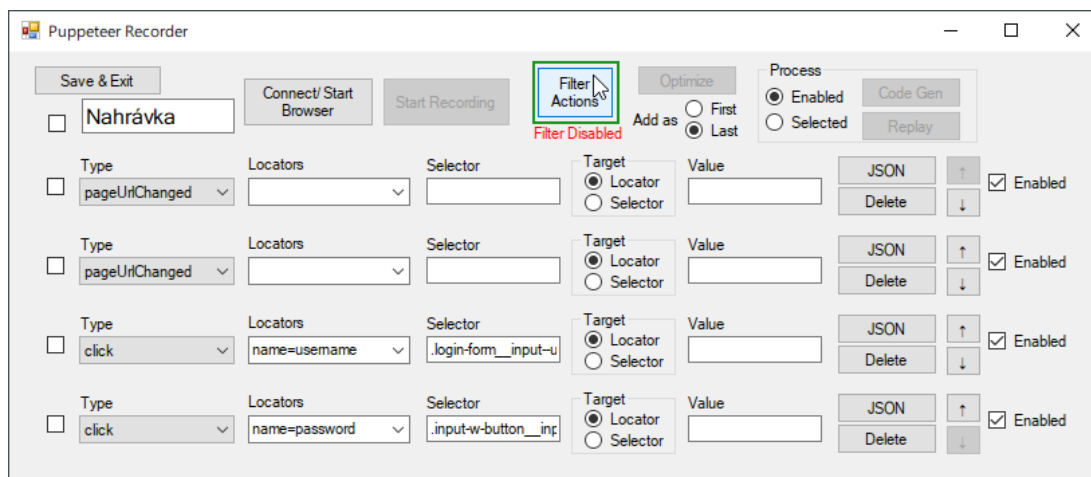


Obrázek 45: Vygenerování kódu jedné akce typu „pageUrlChanged“

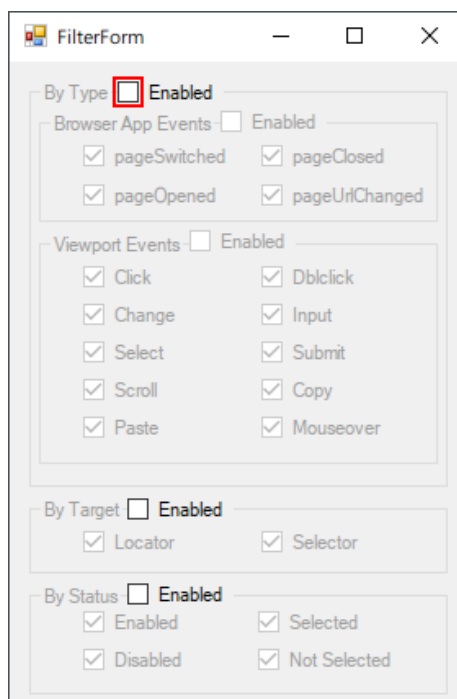
4.8.6 Filtr akcí

V horní části editačního UI je ještě jedno tlačítko, na které jsme se zatím v žádném tutoriálu nepodívali. Tlačítko má popisek „Filter Actions“, který nám napovídá, že jedná se o filtr. V našem případě umožňuje dočasně skrýt akce dle různých kritérií¹.

Přesuňme se do existující nahrávky a klikněme na tlačítko pro filtraci.



(a) Klik na tlačítko „Filter Actions“



(b) Okno pro aplikaci filtrů

Obrázek 46: Otevření okna pro aplikaci filtrů

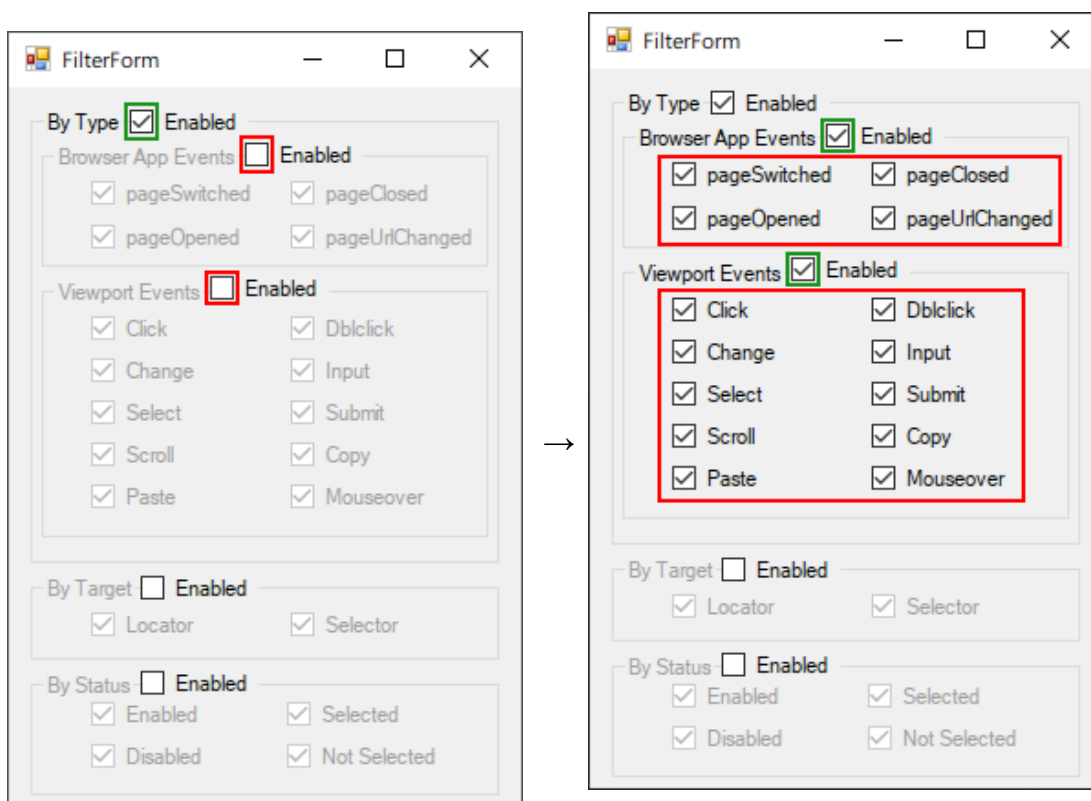
¹ Kritéria, která můžeme použít pro filtrování (i současně):

- typ události (pageUriChanged, click, scroll, ...)
- typ identifikátoru (lokátor, selektor)
- stav akce (povolena, vybraná)

Všimněme si, že na obr. 46(a) je pod stisknutým tlačítkem napsáno „Filter Disabled“, což značí neaktivitu filtru. Dále se podívejme, že na stejném obrázku se mezi čtyřmi nahranými akcemi nachází nejprve dvakrát za sebou akce typu „pageUrlChanged“ a potom dvakrát za sebou akce typu „click“. Filtr využijeme tak, že skryjeme poslední dvě akce.

Pro použití filtru nejprve musíme aktivovat filtr odpovídající kategorie. To se provádí zaškrtnutím políčka s popiskem „Enabled“. Políčko pro aktivaci filtru dle typu akce je zvýrazněno červeným rámečkem na obr. 46(b).

Tuto aktivaci filtru provedeme, poté zaškrtneme políčko s popiskem „Browser App Events“ a „Viewport Events“. První zmíněné políčko povolí filtrování akcí pocházejících z okna prohlížeče, druhé povolí filtrování akcí z viewportu.



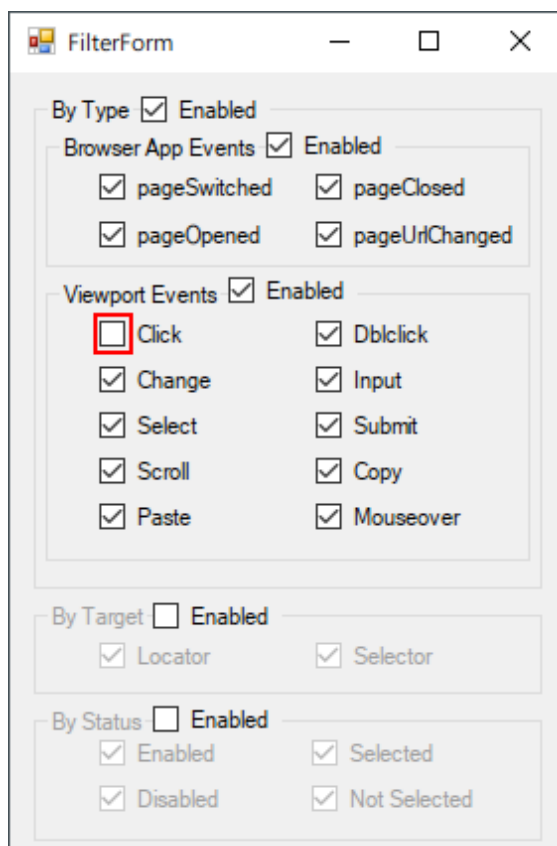
(a) Aktivace filtru dle typu akce

(b) Povolení filtrování akcí pocházejících z okna prohlížeče i z viewportu

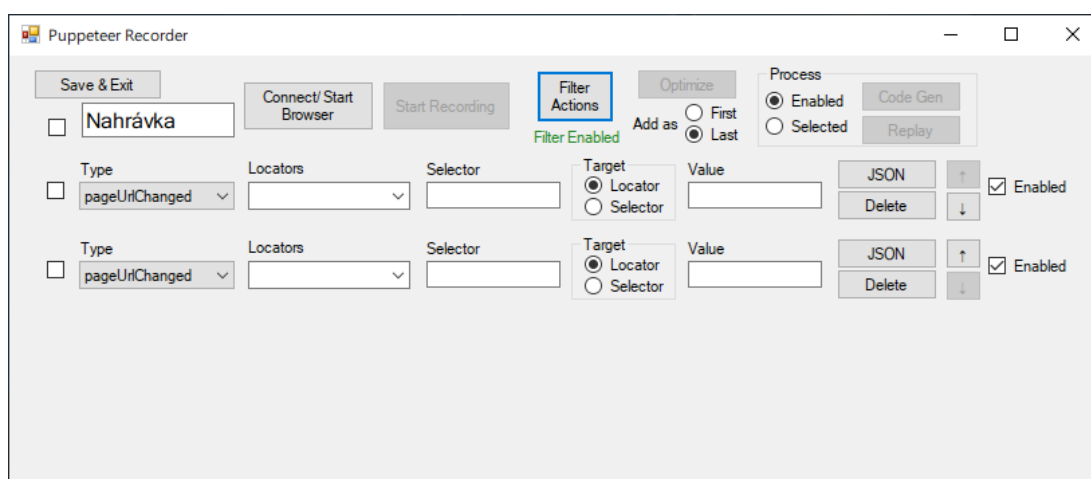
Obrázek 47: Aktivace filtru pro akce pocházející z okna prohlížeče a viewportu

Zaškrtnutá políčka, která jsou obsažena v červeném rámečku na obr. 47(b) značí typy akcí, které budou zobrazeny v editačním UI.

Pro skrytí posledních dvou akcí typu „click“ stačí pouze odškrtnout políčko s popiskem „Click“.



(a) Odškrtnutí políčka s popiskem „Click“



(b) Editační UI se zapnutým filtrem

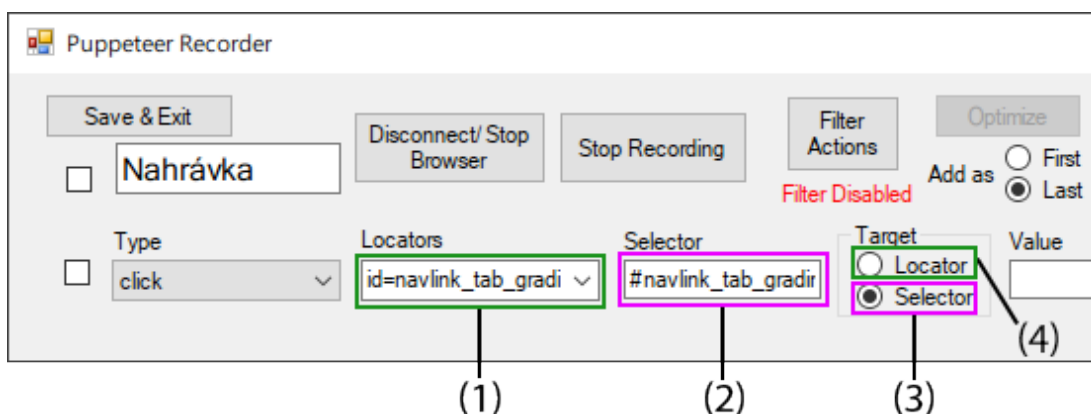
Obrázek 48: Skrytí dvou akcí typu „click“ pomocí filtru

Všimněme si, že na obr. 48(b) se nyní pod tlačítkem s popiskem „Filter Actions“ nachází text zelené barvy „Filter Enabled“ informující o zapnutí filtru.

4.8.7 Změna identifikátoru akce

Akce pocházející z viewportu prohlížeče jsou vždy vykonány nad elementem, který je potřeba identifikovat. K tomu nám slouží vhodný selektor, nebo lokátor. V tomto tutoriálu si ukážeme jak pracovat se selektory a lokátory.

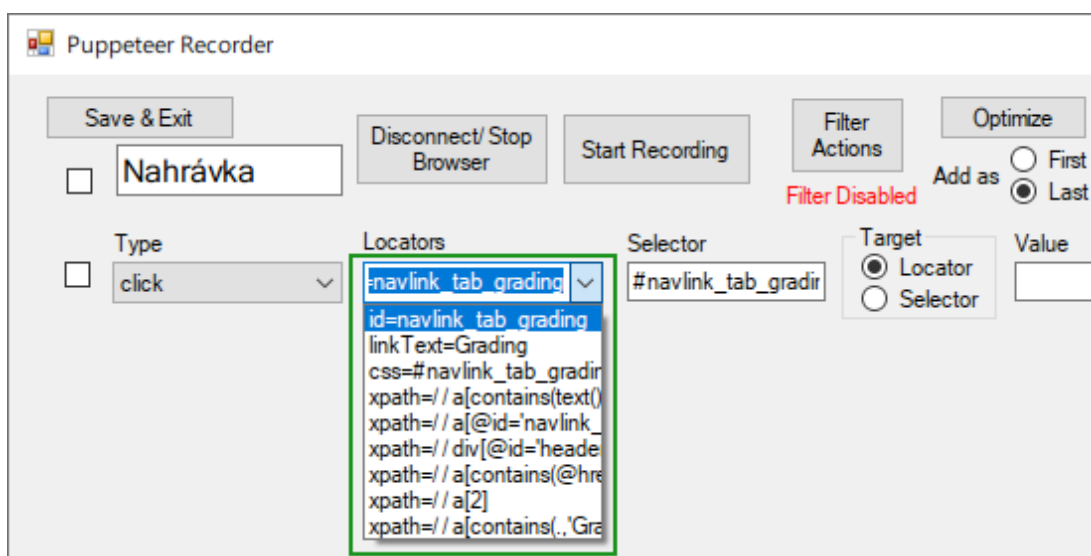
Přesuneme se opět do editačního UI existující nahrávky. Barvy rámečků na následujícím obrázku zobrazují vztahy mezi přepínači a políčky.



Obrázek 49: Vztahy mezi přepínači a políčky

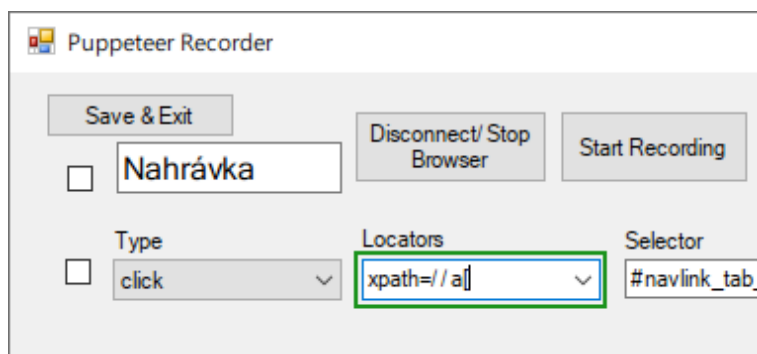
Na obr. 49 je vybrán přepínač (3), jako identifikátor se bude brát v potaz pouze hodnota selektoru (2).

Pokud bychom naopak chtěli použít lokátor jako identifikátor, stačí aby byl vybrán přepínač (4). V tomto případě se bude ignorovat hodnota (2) a použije se vybraný lokátor (1), povšimněme si, že se jedná o combo box, protože obvykle existuje více různých lokátorů identifikujících element.



Obrázek 50: Seznam dostupných lokátorů pro konkrétní akci typu „click“

Pokud bychom chtěli použít vlastní lokátor, který není v seznamu mezi dostupnými, můžeme přepsat hodnotu combo boxu na libovolný vlastní lokátor.

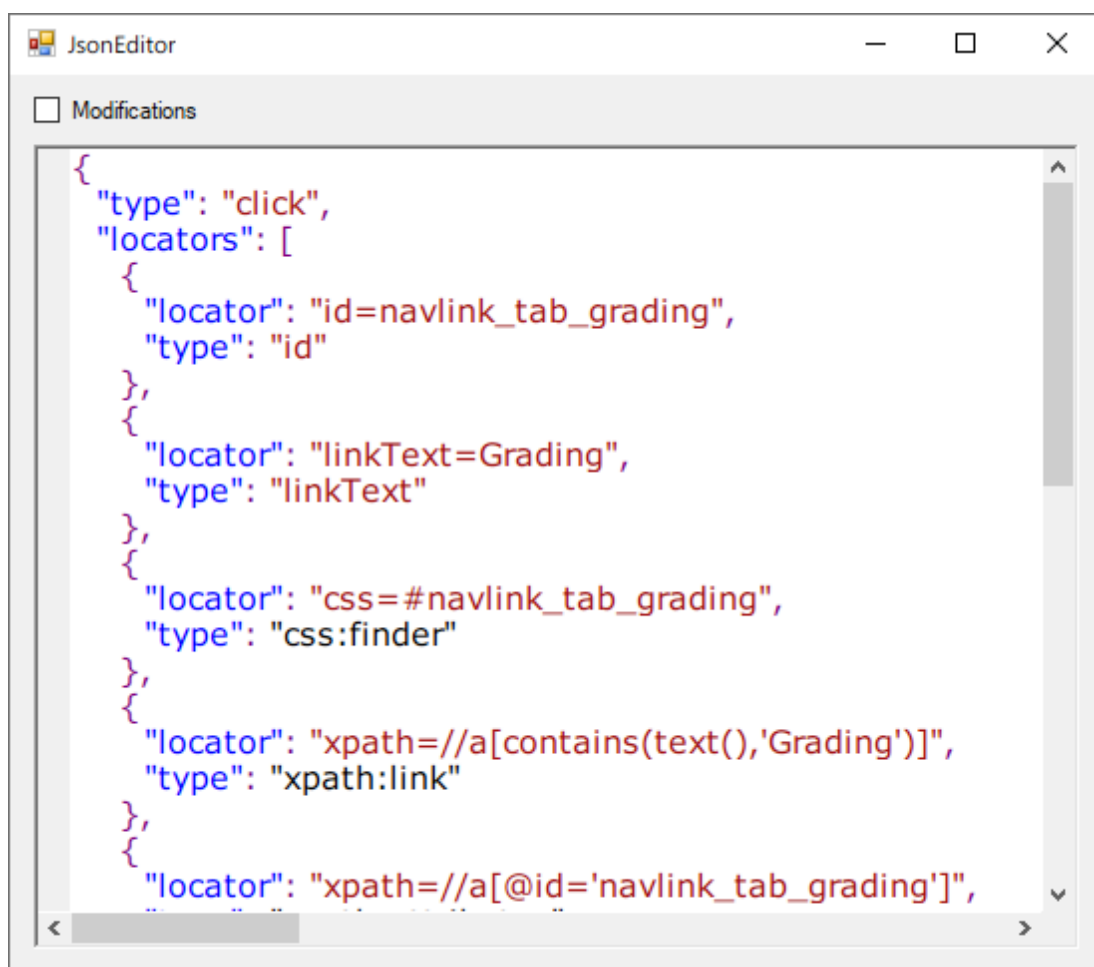


Obrázek 51: Vlastní lokátor

Seznam všech různých typů lokátorů je uveden v tab. 2 na straně 9.

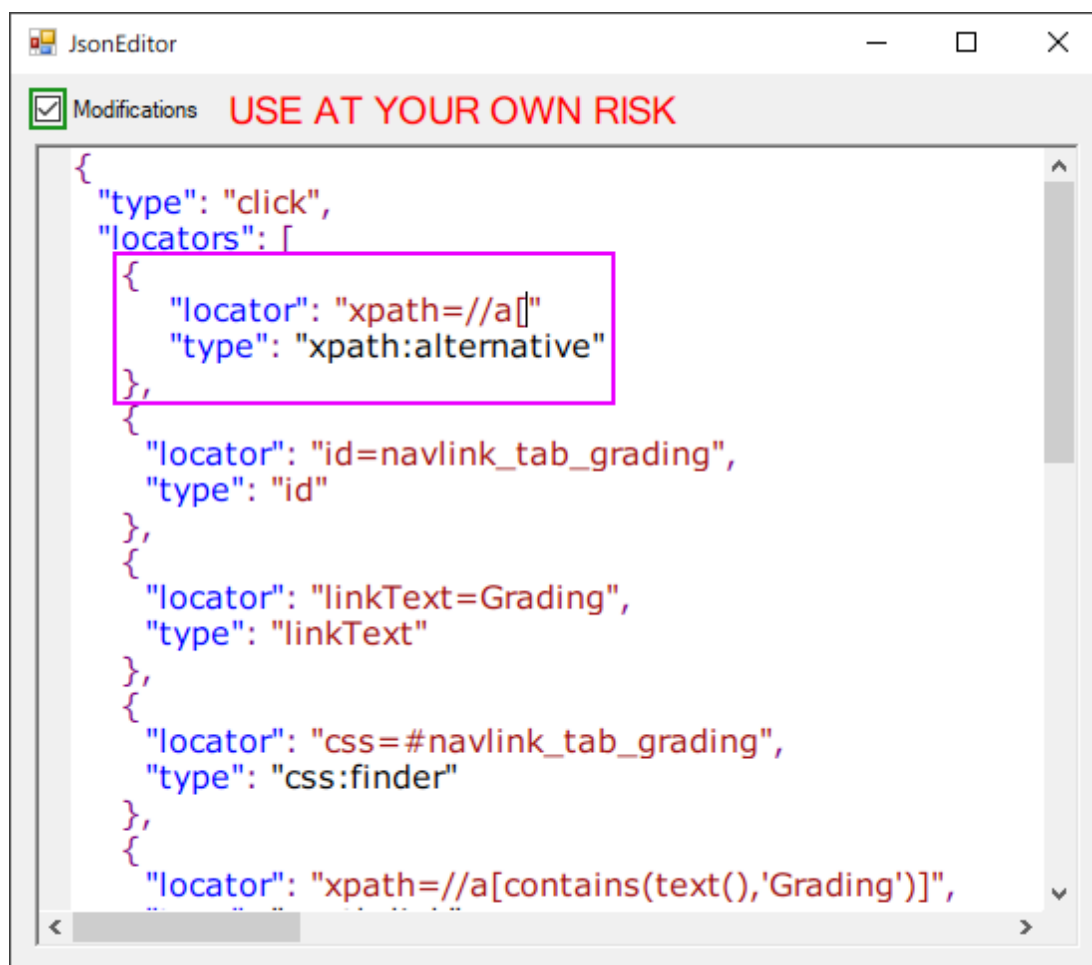
Čtenáře, který pečlivě četl tutoriál 4.8.1 Prvotní spuštění, jistě napadne, že další způsob přidání i většího množství vlastních lokátorů je pomocí tlačítka „JSON“.

Data ve formátu JSON popisující akci typu „click“ vypadají takto:



Obrázek 52: Data ve formátu JSON akce typu „click“

Po povolení modifikací můžeme přidat další lokátory.



Obrázek 53: Přidání nového lokátoru

4.9 Programátorská dokumentace

V této podsekci se budeme zabývat podrobněji backendem a frontendem. Nebudeme se zde zabývat návrhem celého řešení, ten je popsán ve 4.2 Design, kam odkážeme případné zájemce.

Součástí nebudou detailní rozборы zdrojových kódů a dalších jiných technických aspektů. Tato sekce slouží zejména jako rozcestník, zájemci po jejím přečtení budou schopni nalézt zdrojový soubor s hledanou funkcionalitou. Pro snažší pochopení jsou zdrojové soubory komentovány vždy na úrovni tříd a většiny metod, zejména pak těch netriviálních.

4.9.1 Backend

Backendová část se nachází v adresáři Backend\ . Názvy podsekcí spadajících pod tuto sekci odpovídají zdrojovým souborům, a proto předpokládáme, že se nacházíme v cestě Backend\src. Zdrojové soubory jsou uvedeny v abecedním pořadí počínaje soubory nacházejícími se v podadresářích.

4.9.1.1 JsExtensions\ArrayExtensions.js

Jednoduché metody rozšiřující Array objekt JavaScriptu.

Třídy

X

Vybrané metody

```
Object.assign(Array.prototype, {  
  removeAt(index) { this.splice(index, 1) }  
})
```

4.9.1.2 JsExtensions\ObjectExtensions.js

Jednoduché metody rozšiřující Object objekt JavaScriptu.

Třídy

X

Vybrané metody

```
Object.assign(Object.prototype, {  
  empty() { return Object.keys(this).length === 0 }  
})
```

4.9.1.3 JsExtensions\StringExtensions.js

Jednoduché metody rozšiřující `String` objekt JavaScriptu.

Třídy

✕

Vybrané metody

```
Object.assign(String.prototype, {  
  containsNumber() { return /^-?\d+$/.test(this) }  
})
```

4.9.1.4 PuppeteerExtensions\BrowserExtensions.js

Několik metod rozšiřujících standardní funkcionalitu `Browser` objektu Puppeteeru. Součástí je i kód redefinující zejména událost „tabcreated“, toho je využíváno pro inicializaci rozšíření nově otevřených stránek viz. 4.9.1.5.

Třídy

✕

Vybrané metody

async function `addBrowserExtensions(browser)` přidá všechna rozšíření do prohlížeče `browser`.

async function `_getActivePage(browser, timeoutMs)` vrátí aktuálně aktivní tab patřící prohlížeči `browser`. Tato metoda nepoběží déle než `timeoutMs`.

4.9.1.5 PuppeteerExtensions\PageExtensions.js

Několik metod rozšiřujících standardní funkcionalitu `Page` objektu Puppeteeru.

Třídy

✕

Vybrané metody

async function `addPageExtensions(page)` přidá všechna rozšíření do stránky `page`.

async function `_cdpExecute(page, cmds)` odešle požadavek stránce `page` na volání CDP funkce dané `cmds`.

function `_clearExposedFunctionsBindings`(page) odstraní všechny funkce, které byly dříve zveřejněny na stránce page.

function `_removeExposedFunctionByName`(page, fName) odstraní funkci se jménem fName, která byla dříve zveřejněna na stránce page.

4.9.1.6 selenium-ide-src\LocatorPkg.js

Funkcionalita lokátorů pochází ze zdrojového kódu projektu Selenium IDE [23]. Na tento kód se vztahuje licence Apache 2.0 v souladu se kterou je kód využit.

Třídy

Jedná se sice o velké množství kódu, většina ale pochází z doby, kdy JavaScript neobsahoval klíčové slovo `class`. Pokud budeme brát v potaz objekty vytvořené přes funkce, jak to bývalo tradiční v JavaScriptu, pak do tohoto seznamu můžeme zahrnout `LocatorBuilders`.

Vybrané metody

`LocatorBuilders.prototype.buildAll` = **function**(el) vrátí seznam lokátorů identifikující el, což je objekt typu `Element` odpovídající nějakému existujícímu elementu na stránce.

4.9.1.7 WindowFunctions\LocatorTransformation.js

Metody pro transformaci nezpracovaných lokátorů získaných pomocí funkce `buildAll` viz 4.9.1.6.

Třídy

x

Vybrané metody

function `transformLocators`(locators) vyextrahuje CSS lokátor z pole `locators`, vrátí dvě hodnoty: pole zbylých lokátorů a CSS lokátor.

4.9.1.8 BrowserConnectionLayer.js

Abstrakční vrstva pro inicializaci Puppeteeru umožňující propojení s prohlížečem.

Třídy

`BrowserConnectionLayer`

Vybrané metody

async `cdpExecuteAllPages(cmds)` okamžitě provede volání CDP funkcí daných parametrem `cmds` na všech otevřených stránkách.

async `initializeBrowser()` přidá do objektu prohlížeče rozšíření z 4.9.1.4.

async `launch(options)` připojí Puppeteer k prohlížeči voláním `puppeteer.launch(options)`. Následně přidá do objektu prohlížeče rozšíření z 4.9.1.4.

`setNewPageCDPExecute(cmds)` nastaví volání CDP funkcí daných parametrem `cmds` pro každý nově otevřený tab.

4.9.1.9 CodeGenerator.js

Logika generování kódu pro Puppeteer.

Třídy

`CodeGenerator`

Vybrané metody

`codegen(actions)` vygeneruje kód akce/akcí v proměnné `actions`.

`_addPermissionToCurrPage(permission)` přidá požadavek na oprávnění `permission` pro aktuální URL aktivní stránky.

`_catchAndLogErrors(code)` vygeneruje kód pro logování a zachytávání výjimek podle hodnot v `this._options.catchErrors` a `this._options.logErrors`.

`_codegenBlock(actions, i)` generuje blok kódu z `actions[i]`. Vygenerovaný blok se skládá ze tří částí: prologue, main a epilogue.

`_genDOMEventFromIdentifier(id, eventName)` generuje volání `Page.evaluate`, nebo `ElementHandle.evaluate` obsahující událost DOM. Typ volání a element aktivující událost jsou určeny na základě identifikátoru `id`, název události je dán parametrem `eventName`.

`_genGivePermissions()` vygeneruje kód pro získání oprávnění na základě předchozích volání metody `_addPermissionToCurrPage`.

`_genWaitForIdentifier(id)` vygeneruje volání `Page.waitForXPath`, nebo `Page.waitForSelector` dle typu identifikátoru `id`.

`_genWaitForNavCall()` vygeneruje volání `Page.waitForNavigation`.

`_identifier(action)` vrátí identifikátor akce `action`, který byl vybrán ve frontendu.

`_plainCode(codeBlocks)` převede bloky kódu vygenerované metodou `_codeGenBlock` do kódu pro Puppeteer.

Mnoho dalších metod generujících volání:

`_browserFuncCall`, `_genExecRawCodeInsidePage`, `_genIdentifierFuncCall`, `_pageFuncCall`, ... Více informací o těchto a dalších metodách je uvedeno v komentářích ve zdrojovém souboru.

4.9.1.10 main.js

Spouštějící soubor backendové části, čeká na zprávy z backendu, které předá ke zpracování.

Třídy

X

Vybrané metody

function `processCmd(cmd)` zpracuje příkaz `cmd` pocházející z frontendu.

4.9.1.11 Optimizer.js

Třídy

`Optimizer`

Vybrané metody

`optimizeRecordings(rec)` smaže nevýznamné, nadbytečné akce pole `rec`, navrátí nové pole bez těchto akcí.

4.9.1.12 Recorder.js

Logika nahrávání akcí pocházejících z okna prohlížeče i viewportu.

Třídy

Recorder, dědí od třídy BrowserConnectionLayer viz 4.9.1.8.

Vybrané metody

async `optimize()` provede optimalizaci nahraných akcí viz 4.9.1.11.

`setEventsToRecord(events)` nastaví události k nahrávání podle `events`.

async `start(cleanStart = true)` spustí nahrávání akcí, `cleanStart` udává jestli má být pole nahraných akcí vyčištěno.

async `stop()` zastaví probíhající nahrávání akcí.

async `_connectViewportEvents(page)` spustí na stránce `page` kód pro zachytávání a předávání akcí zpátky do backendové aplikace.

4.9.1.13 WindowFunctionsUtils.js

Metody umožňující zveřejnit obsah zdrojového souboru do proměnné `window` okna prohlížeče.

Třídy

X

Vybrané metody

async function `addWindowFunctionsToPage(page)` přidá do proměnné `window` stránky `page` obsah souborů 4.9.1.6 a 4.9.1.7.

4.9.1.14 ZeroMQUtils.js

Několik metod usnadňujících práci se ZeroMQ.js.

Třídy

X

Vybrané metody

```
async function receiveMsg(sock) {  
  return (await sock.receive()).toString('utf-8')  
}
```

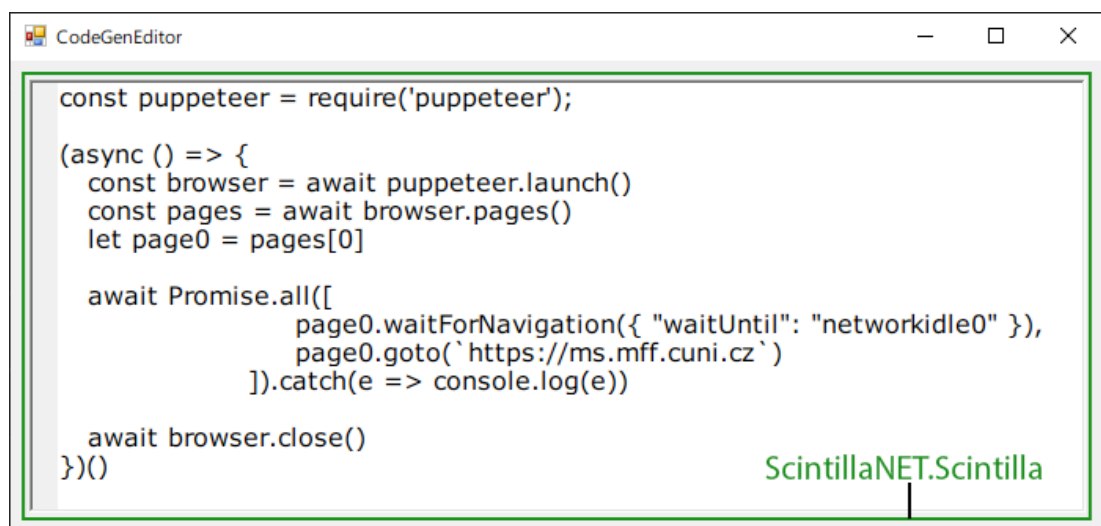

4.9.2 Frontend

Visual Studio projekt frontendové části se nachází v adresáři `Frontend\`. Zdrojový kód pak v adresáři `Frontend\Frontend`, který považujeme nyní za aktuální cestu. Stejně jako v předchozím 4.9.1 Backend, i zde názvy podsekcí odpovídají názvům zdrojových souborů, které jsou uvedeny v abecedním pořadí.

Upřesníme navíc ještě, že všechny formuláře se nacházejí v adresáři `Forms\` a uživatelské komponenty (user control) využívané ve formulářích pak v `UserControls\`.

4.9.2.1 Forms\CodeGenEditor.cs

Textový editor zobrazující vygenerovaný kód pro Puppeteer. Použitý textový editor je komponenta `ScintillaNET.Scintilla` projektu `ScintillaNET` [25] dostupného pod MIT licencí v souladu se kterou je komponenta využita.



Obrázek 54: Formulář CodeGenEditor s komponentou `ScintillaNET.Scintilla`

Třídy

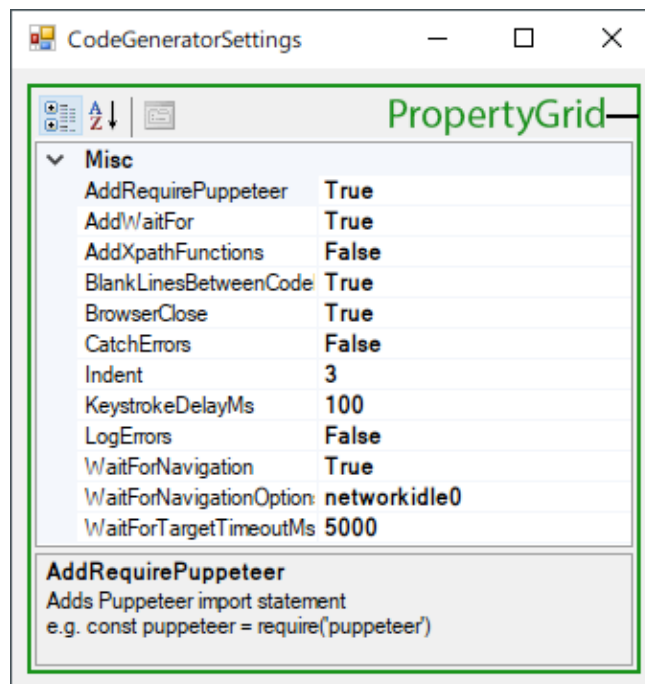
`CodeGenEditor` : `Form`

Vybrané metody

`public void SetEditorText(string txt)` nastaví textový řetězec `txt` jako obsah textového editoru.

4.9.2.2 Forms\CodeGenSettingsForm.cs

Nastavení parametrů pro generování kódu.



Obrázek 55: Formulář CodeGenSettingsForm s komponentou PropertyGrid

Třídy

CodeGenSettingsForm : Form

Vybrané metody

public void BindCodeGeneratorOptions (CodeGenOptions c) nastaví parametr c do komponenty property grid jako objekt k úpravě.

public CodeGenOptions ExportCodeGeneratorOptions () vrátí objekt nastavený dle hodnot v property gridu.

4.9.2.3 Forms\FilterForm.cs

Nastavení filtru.

FilterForm

By Type ☒ Enabled

Browser App Events ☒ Enabled

☒ pageSwitched ☒ pageClosed

☒ pageOpened ☒ pageUrlChanged

Viewport Events ☒ Enabled

☒ Click ☒ Dblclick

☒ Change ☒ Input

☒ Select ☒ Submit

☒ Scroll ☒ Copy

☒ Paste ☒ Mouseover

By Target ☒ Enabled

☒ Locator ☒ Selector

By Status ☒ Enabled

☒ Enabled ☒ Selected

☒ Disabled ☒ Not Selected

Obrázek 56: Formulář FilterForm

Třídy

FilterForm : Form

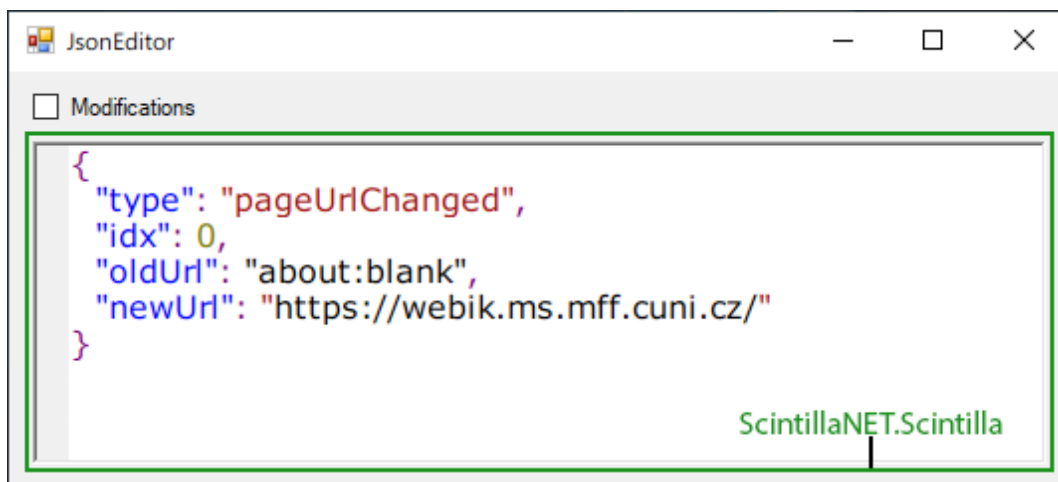
Vybrané metody

public Filter ExportFilter() vrátí objekt odpovídající aktuálnímu formuláři.

public void SetFilter(Filter f) nastaví UI formuláře podle parametru f.

4.9.2.4 Forms\JsonEditor.cs

Textový prohlížeč a editor nezpracovaného JSONu odpovídající akci. Použitý textový editor je komponenta `ScintillaNET.Scintilla` projektu `ScintillaNET` [25] dostupného pod MIT licencí v souladu se kterou je komponenta využita.



Obrázek 57: Formulář `JSONEditor` s komponentou `ScintillaNET.Scintilla`

Třídy

`JsonEditor` : `Form`

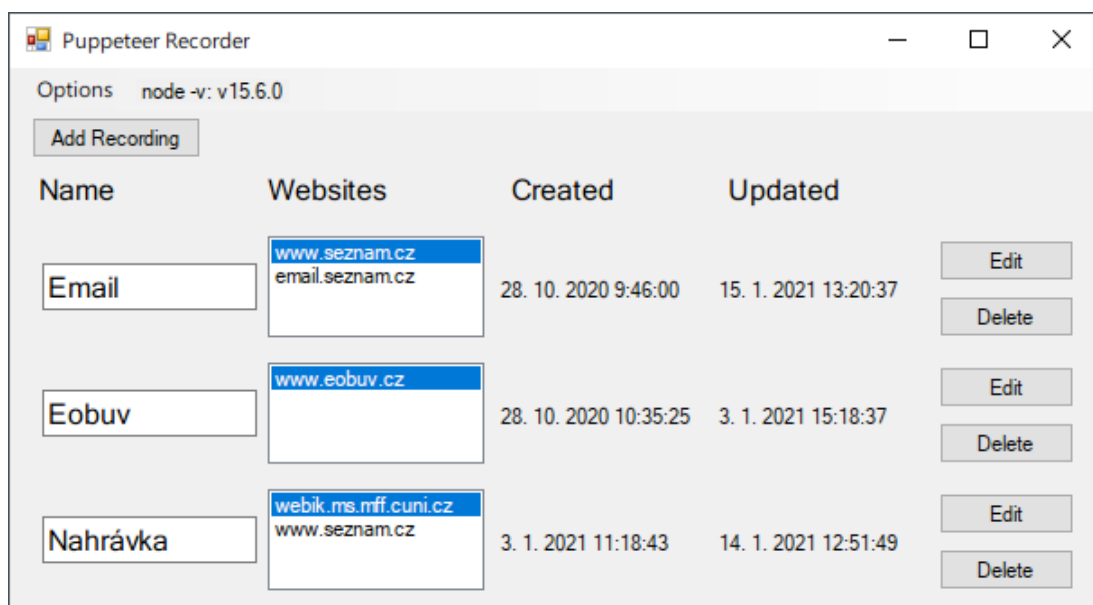
Vybrané metody

`public string` `GetEditorText()` vrátí obsah textového editoru.

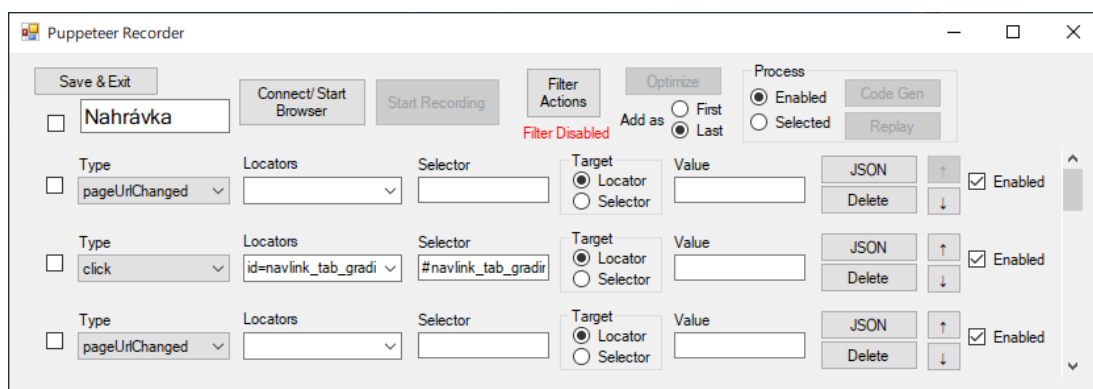
`public void` `SetEditorText(string txt)` nastaví textový řetězec `txt` jako obsah textového editoru. Hodnota vlastnosti `ReadOnly` textového editoru zůstane zachována.

4.9.2.5 Forms\MainForm.cs

Hlavní spouštějící formulář frontendu, nachází se v jednom ze dvou stavů – po spuštění je vždy ve stavu `AppMode.List`, kdy zobrazuje náhledy všech uložených nahrávek. Při vytvoření nové nahrávky nebo editaci existující se stav přepne do `AppMode.Edit`. Tyto dva stavy zobrazuje následující obrázek:



(a) `AppMode.List`



(b) `AppMode.Edit`

Obrázek 58: Stavy `AppMode.List` a `AppMode.Edit` formuláře `MainForm`

Třidy

`MainForm` : `Form`

Vybrané metody

private void MainForm_Load(**object** sender, EventArgs e) spuštěna při načtení formuláře, načte uložené náhledy nahrávek a zkontroluje existenci interpreteru Node.js.

protected override bool ProcessCmdKey(**ref** Message msg, Keys keyData) přetížení definující klávesovou zkratku Shift+Delete při editaci nahrávky. Tato zkratka odstraní akci s vlastností `Focus = true`.

public void SwitchToEditMode(Thumbnail t) přepne formulář do stavu `AppMode.Edit` zobrazeném na obr. 58(b). Parametr `t` je náhled editované nahrávky. V případě, že `t` není náhled nově vytvořené nahrávky, dříve uložené nahrané akce jsou načteny do formuláře.

Další metody zpracovávající události UI:

`addNewRecordingButton_Click`, `MainForm_FormClosing`,
`recorderToolStripMenuItem_Click`, ...

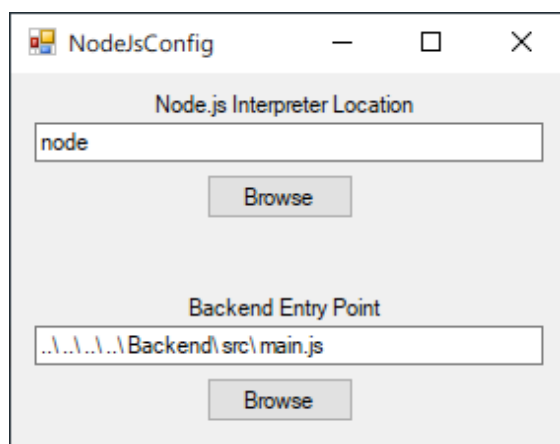
A metody upravující UI:

`LoadThumbnailsUi`, `NodeInterpreterNotWorking`, `PerformSettingsChecks`,
`SetEditUiVisiblity`, `SwitchToEditMode`, `UiChange` ...

Více informací o těchto a všech dalších metodách je uvedeno v komentářích ve zdrojovém souboru.

4.9.2.6 Forms\NodeJsConfig.cs

Nastavení cesty k backendu a interpreteru Node.js.



Obrázek 59: Formulář NodeJsConfig

Třidy

`NodeJsConfig` : `Form`

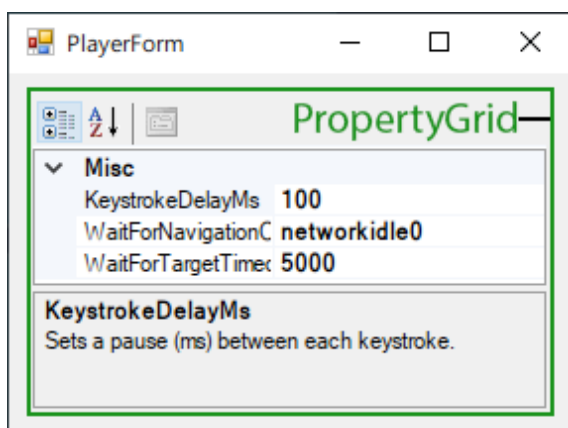
Vybrané metody

public void BindNodeJsOptions(NodeJsOptions n) nastaví UI formuláře podle parametru n.

public NodeJsOptions ExportNodeJsOptions() vrátí objekt odpovídající aktuálnímu formuláři.

4.9.2.7 Forms\PlayerForm.cs

Nastavení parametrů pro přehrávání akcí.



Obrázek 60: Formulář PlayerForm s komponentou PropertyGrid

Třídy

PlayerForm : Form

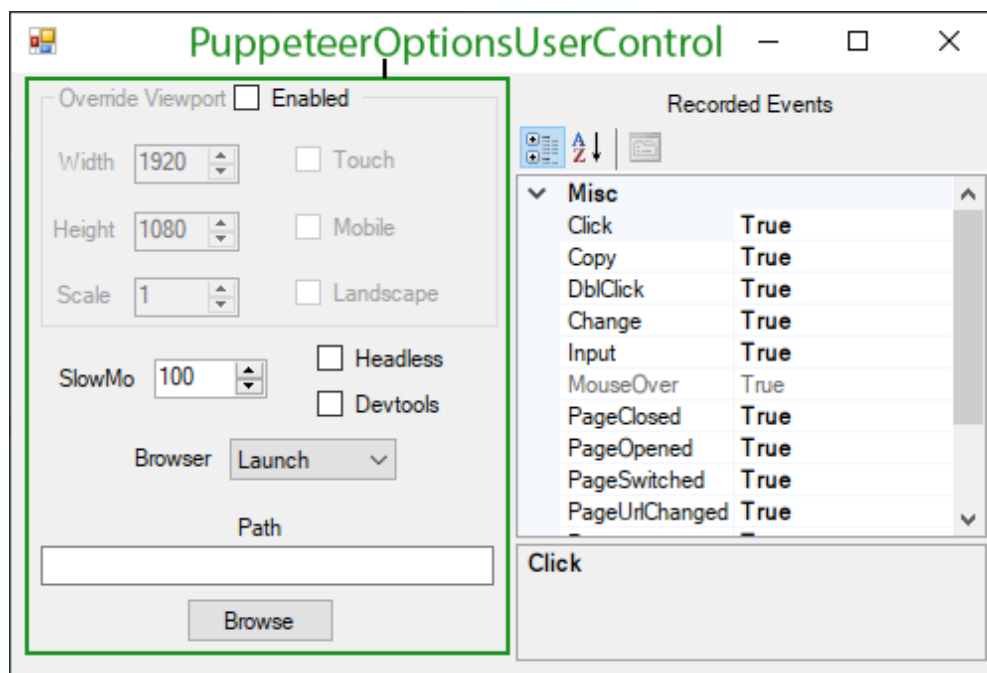
Vybrané metody

public void BindNodeJsOptions(NodeJsOptions n) nastaví parametr n do komponenty property grid jako objekt k úpravě.

public NodeJsOptions ExportNodeJsOptions() vrátí objekt nastavený dle hodnot v property gridu.

4.9.2.8 Forms\RecorderSettingsForm.cs

Nastavení nahrávání a prohlížeče.



Obrázek 61: Formulář RecorderSettingsForm s uživatelskou komponentou PuppeteerOptionsUserControl

Třídy

RecorderSettingsForm : Form

Vybrané metody

public RecorderConfiguration ExportRecorderOptions() vrátí objekt nastavený dle aktuálních hodnot formuláře.

private void RecorderSettings_Load(object sender, EventArgs e) spuštěna při načtení formuláře, vyplní UI dle uloženého nastavení.

4.9.2.9 Forms\ReplayViewForm.cs

Stav, informace a chyby aktuálně probíhajícího přehrávání.



Obrázek 62: Formulář ReplayView ve stavu ukončeného přehrávání s jednou chybou

Třídy

ReplayViewForm : Form

Vybrané metody

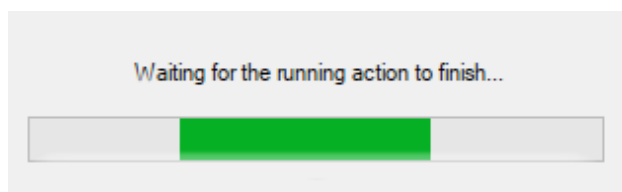
`public void AddError(string errorMsg, int id)` přidá chybu errorMsg do data grid view a propojí ji s ActionUserControl (4.9.2.11) podle id.

`public void SetRecordingEnded(bool state)` podle parametru state nastaví formulář do stavu probíhajícího, nebo ukončeného přehrávání.

Mnoho dalších metod zpracovávajících události UI: dataGridView_CellClick, ReplayViewForm_FormClosing, stopButton_Click, ... Více informací o těchto a dalších metodách je uvedeno v komentářích ve zdrojovém souboru.

4.9.2.10 Forms\WaitingWindow.cs

Formulář vyzývající k čekání.



Obrázek 63: Formulář WaitingWindow

Třídy

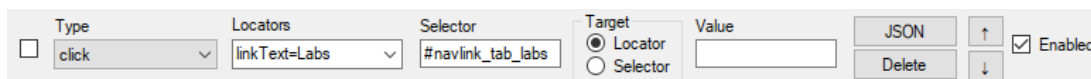
WaitingWindow : Form

Vybrané metody

X

4.9.2.11 UserControl\ActionUserControl.cs

UI pro 4.9.2.28 zobrazené v jiné uživatelské komponentě 4.9.2.12.



Obrázek 64: Uživatelská komponenta ActionUserControl pro akci typu „click“

Třídy

ActionUserControl : UserControl

Vybrané metody

public void BindAction(dynamic a, **int** id) nastaví akci a do uživatelské komponenty, kterou obnoví do výchozího stavu. Nastavenou akci bude možné identifikovat pomocí id.

public void BindRecording(Recording r, **int** id) nastaví akci a UI z parametru r. Nastavenou akci bude možné identifikovat pomocí id.

public dynamic ExportActionForOutput() vrátí akci v JSONu dle aktuální komponenty. Objekt v JSONu bude navíc obsahovat parametr „target“ definující vybraný identifikátor.

public Recording ExportRecordingForSave() vrátí objekt odpovídající aktuální komponentě.

Metody obsahující logiku prohazování pořadí akcí¹:

downButton_Click, GetNextVisiblePosition, GetPreviousVisiblePosition, UpdateUpDownButtons, upButton_Click.

A metody zpracovávající další události UI:

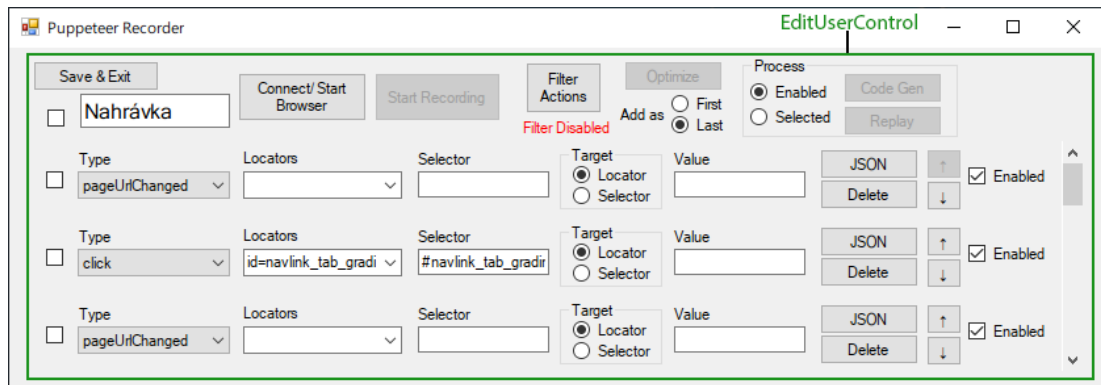
deleteButton_Click, jsonEditButton_Click, selectorTextBox_TextChanged, typeComboBox_SelectedValueChanged, valueTextBox_TextChanged, ...

Více informací o těchto a všech dalších metodách je uvedeno v komentářích ve zdrojovém souboru.

¹ Tyto metody jsou využity při stisku tlačítek  a  nacházejících se na obr. 64.

4.9.2.12 UserControl\EditUserControl.cs

Uživatelská komponenta pro práci s nahrávkou: nahrávání, přehrávání, generování kódu,... Pokud se hlavní okno 4.9.2.5 nachází ve stavu `AppMode.Edit`, tato uživatelská komponenta vyplňuje celé okno.



Obrázek 65: Uživatelská komponenta `EditUserControl` jako součást formuláře `MainWindow`

Třídy

`EditUserControl : UserControl`

Vybrané metody

`private void AddErrorToReplayViewForm(string msg, int id)` přidá chybu `msg` do seznamu chyb, které nastaly při přehrávání. Přidaná chyba je propojena s `ActionUserControl` viz 4.9.2.11 pomocí parametru `id`. Při kliku na chybu je `ActionUserControl` zvýrazněna červenou barvou.

`public void BindEdit(CurrentEdit ce)` naváže nahrávku `ce` do uživatelské komponenty.

`private void FinishReplay()` informuje backend o odeslání poslední akce k přehrávání, poté čeká na potvrzení o dokončení přehrávání poslední akce.

`private List<Recording> GetAllActions()` vrátí všechny akce včetně jejich `id` a nastavení uživatelské komponenty `ActionUserControl` viz 4.9.2.11.


`private List<string> GetEventsToRecord()` vrátí jména událostí povolených k nahrávání.

`private List<Recording> GetRecordingsForOptimize()` vrátí akce určené k následné optimalizaci.

private void LoadAction(dynamic action) pro akci action vytvoří uživatelskou komponentu `ActionUserControl` (4.9.2.11), kterou přidá do seznamu nahraných akcí. Nově přidaná komponenta má výchozí UI.

private void LoadRecording(Recording r) pro akci r vytvoří uživatelskou komponentu `ActionUserControl` (4.9.2.11), kterou přidá do seznamu nahraných akcí. Nově přidaná komponenta má UI nastavené podle vlastnosti `r.UiConfig`.

private void RecordingTask() naslouchá na informace z backendu o zaznamenaných akcích. Po obdržení akce ji přidá do seznamu akcí voláním metody `LoadAction`.

private void ReplayTask() postupně odesílá akce k přehrání do backendu, `ActionUserControl` (4.9.2.11) aktuálně přehrávané akce je zvýrazněna  barvou.

private `HashSet<string>` ScrapeWebsitesFromRecordings(List<Recording> recordings) vrátí seznam načtených adres URL z akcí recordings.

private void StartNodeJsProcess() spustí backend.

Další metody zpracovávající události UI: `browserConnection_Click`, `processRadioButtons_CheckedChanged`, `saveAndExitButton_Click`, ...

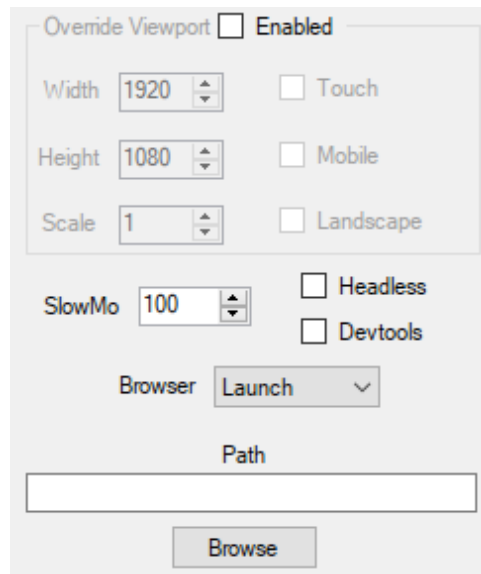
A metody upravující UI:

`ActionUserControlCheckedChanged`, `ClearErrorCustomColors`, `FilterChanged`, `HighlightActionUserControlById`, `SetReplayEndedVisibility`, `SomeActionsSelectedForProcessing`, `UpdateAllActionUpDownButtons`, `UpdateUi`, ...

Více informací o těchto a všech dalších metodách je uvedeno v komentářích ve zdrojovém souboru.

4.9.2.13 UserControls\PuppeteerOptionsUserControl.cs

Nastavení parametrů připojení k prohlížeči, tato uživatelská komponenta je využita ve formuláři 4.9.2.8.



Obrázek 66: Uživatelská komponenta PuppeteerOptionsUserControl

Třídy

PuppeteerOptionsUserControl : UserControl

Vybrané metody

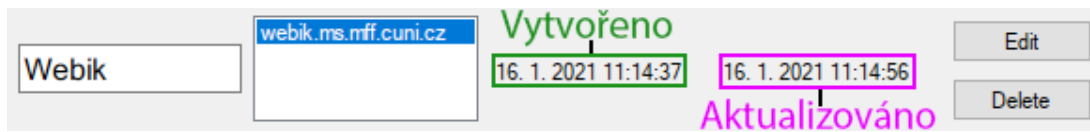
public void BindOptions(PuppeteerOptions opts) nastaví UI komponenty podle parametru opts.

public PuppeteerOptions ExportOptions() vrátí objekt odpovídající aktuální komponentě.

private void SetBrowserMode(BrowserMode mode) podle parametru mode nabývající hodnoty Launch, nebo Connect nastaví UI pro vyplnění cesty, nebo IP adresy s portem.

4.9.2.14 UserControl\ThumbnailUserControl.cs

Náhled na uloženou nahrávku.



Obrázek 67: Uživatelská komponenta ThumbnailUserControl zobrazující náhled nahrávky s názvem „Webik“ provedenou na stránce webik.ms.mff.cuni.cz

Třídy

ThumbnailUserControl : UserControl

Vybrané metody

public void BindThumbnail(Thumbnail t) nastaví UI komponenty podle parametru t.

4.9.2.15 CodeGenOptions.cs

Parametry pro nastavení generátoru kódu.

Třídy

CodeGenOptions

Vybrané metody

X

4.9.2.16 ConfigManager.cs

Nastavení, jeho načtení a uložení, v případě, že neexistuje i generování výchozího.

Třídy

ConfigManager

Vybrané metody

public static CodeGenOptions GetCodeGeneratorOptions() vrátí uložené nastavení (výchozí pokud neexistuje) generátoru kódu.

public static NodeJsOptions GetNodeJsOptions() vrátí uložené nastavení (výchozí pokud neexistuje) obsahující cestu k backendu a interpreteru Node.js.

public static PlayerOptions GetPlayerOptions() vrátí uložené nastavení (výchozí pokud neexistuje) pro přehrávání akcí.

public static PuppeteerOptions GetPuppeteerConfiguration() vrátí uložené nastavení (výchozí pokud neexistuje) uživatelské komponenty 4.9.2.13.

public static RecordedEvents GetRecordedEventsOptions() vrátí uložené nastavení (výchozí pokud neexistuje) obsahující typy akcí povolené k nahrávání.

public static void SavePuppeteerConfiguration(PuppeteerOptions po) uloží nastavení uživatelské komponenty 4.9.2.13.

public static void SaveCodeGeneratorOptions(CodeGenOptions cgo) uloží nastavení generátoru kódu.

public static void SaveNodeJsOptions(NodeJsOptions no) uloží nastavení obsahující cestu k backendu a interpreteru Node.js.

public static void SavePlayerOptions(PlayerOptions po) uloží nastavení přehrávání akcí.

public static void SaveRecordedEventsConfiguration(RecordedEvents po) uloží nastavení obsahující typy akcí povolené k nahrávání.

4.9.2.17 Configuration.cs

Reprezentace celkového nastavení frontendu.

Třídy

Configuration

Vybrané metody

X

4.9.2.18 Constants.cs

Konstantní cesty k souborům obsahujícím nastavení.

Třídy

Constants

Vybrané metody

X

4.9.2.19 CurrentEdit.cs

Reprezentace nahrávky včetně náhledu (4.9.2.30) a kompletního nastavení (4.9.2.17).

Třídy

`CurrentEdit`

Vybrané metody

X

4.9.2.20 Filter.cs

Reprezentace filtru využitá v kódu formuláře 4.9.2.3.

Třídy

`Filter`

Vybrané metody

X

4.9.2.21 NodeJsOptions.cs

Reprezentace nastavení cesty k backendu a interpreteru Node.js.

Třídy

`NodeJsOptions`

Vybrané metody

X

4.9.2.22 PairSocketExtensions.cs

Rozšíření objektu `PairSocket` knihovny NetMQ.

Třídy

`PairSocketExtensions`

Vybrané metody

`public static bool ReceiveFrameStringTimeout(this PairSocket pair, out string result, int timeoutMs)` čeká na textová data socketu `pair` maximálně `timeoutMs` milisekund.

4.9.2.23 PlayerOptions.cs

Reprezentuje nastavení pro přehrávání akcí.

Třídy

PlayerOptions

Vybrané metody

X

4.9.2.24 Program.cs

Spouštějící soubor frontendové části.

Třídy

Program

Vybrané metody

`private static void Main()` spustí MainForm (4.9.2.5).

4.9.2.25 PuppeteerOptions.cs

Reprezentace nastavení uživatelské komponenty 4.9.2.13.

Třídy

PuppeteerOptions obsahuje pouze nastavení prohlížeče (headless režim, specifický viewport, zobrazit devtools, ...). Součástí není typ připojení k prohlížeči.

ConnectPuppeteerOptions : PuppeteerOptions definuje navíc vlastnost Uri EndPoint obsahující IP adresu a port.

LaunchPuppeteerOptions : PuppeteerOptions definuje navíc vlastnost string ExecutablePath obsahující cestu ke spustitelnému souboru prohlížeče.

Viewport popisuje specifický viewport.

Vybrané metody

X

4.9.2.26 RecordedEvents.cs

Reprezentace nastavení typů akcí povolených k nahrávání.

Třídy

`RecordedEvents`

Vybrané metody

X

4.9.2.27 RecorderConfiguration.cs

Reprezentace nastavení formuláře 4.9.2.8.

Třídy

`RecorderConfiguration`

Vybrané metody

X

4.9.2.28 Recording.cs

Reprezentace nahrané akce s nastavením její uživatelské komponenty `ActionUserControl` (4.9.2.11).

Třídy

`Recording`

`UiConfig`

Vybrané metody

X

4.9.2.29 RecordingManager.cs

Uložení nových nahrávek a změny v existujících.

Třídy

`RecordingManager`

Vybrané metody

public static void SaveCurrentEdit(CurrentEdit edit) uloží nahrávku reprezentovanou parametrem edit.

4.9.2.30 Thumbnail.cs

Reprezentace náhledu uložené nahrávky, tyto náhledy se zobrazují pomocí uživatelské komponenty 4.9.2.14.

Třídy

Thumbnail

Vybrané metody

X

4.9.2.31 ThumbnailManager.cs

Správa náhledů nahrávek: vytváření nových, odstranění a načítání existujících.

Třídy

ThumbnailManager

Vybrané metody

public static List<Thumbnail> Init() načte uložené náhledy nahrávek, pokud soubor s náhledy neexistuje, tak ho vytvoří.

public static List<Thumbnail> LoadThumbnails() načte náhledy ze souboru, o kterém předpokládá, že existuje.

public static Thumbnail NewThumbnail() vytvoří náhled nové nepojmenované nahrávky s aktuálním datem vytvoření.

public static void SaveThumbnail(Thumbnail t) uloží náhled t do souboru všech náhledů.

public static void RemoveThumbnail(Thumbnail t) odstraní náhled t ze souboru všech náhledů.

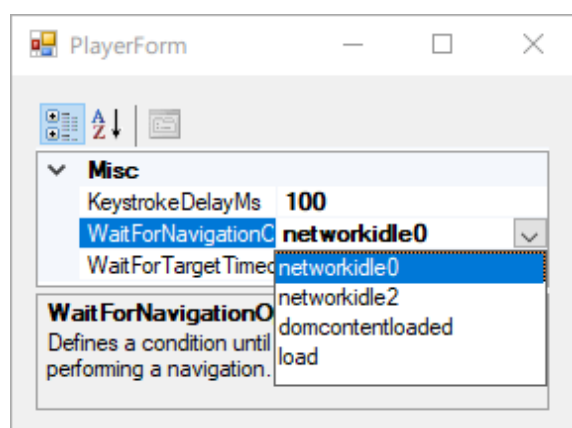
4.9.2.32 WaitForNavigationEnum.cs

Součástí je pouze jeden výčtový typ:

```
public enum WaitForNavigation {  
    networkidle0, networkidle2, domcontentloaded, load  
}
```

Přípustné hodnoty výčtového typu `WaitForNavigation` odpovídají odlišným podmínkám čekání na dokončení navigace, detailně jsou vysvětleny v dokumentaci Puppeteeru [51].

Tento výčtový typ je používán formulářem 4.9.2.7, jak ukazuje následující obrázek:



Obrázek 68: Výčtový typ `WaitForNavigation` jako rozbalovací seznam v property gridu formuláře `PlayerForm`

Třídy

X

Vybrané metody

X

5 Závěr

Tato práce se zabývá automatickým ovládáním webových prohlížečů, hlavním cílem bylo vytvořit řešení využívající knihovnu Puppeteer. Nyní shrneme požadavky, které jsme na naše řešení kladli. V následujícím textu se budeme odvolávat na požadavky uvedené ve 4.1 Požadavky.

Puppeteer se povedlo využít pro nahrávání akcí z okna prohlížeče, řešení umí zachytit otevření nového tabu. Navržen byl ale i způsob pro zaznamenávání akcí z viewportu. Toto odpovídá splnění požadavku (F1).

Při zaznamenávání akce využíváme kód Selenia IDE pro získání lokátorů a selektorů identifikujících element, který akci vyvolal. Tímto jsme splnili požadavek (F4).

Součástí implementace je generátor kódu, který vygeneruje skript pro Puppeteer dle nahraných akcí. Uživatel může skript exportovat a využít ho jako součást své vlastní aplikace. Řešení nabízí ale i možnost pouze přehrát nahrané akce, což se provede jednoduše voláním metody `eval` na vygenerovaný kód. Tímto jsou splněny požadavky (F2)–(F3).

Celé řešení se ovládá z aplikace s GUI vytvořené pomocí WinForms, GUI podporuje škálování na high DPI monitorech, což společně znamená splnění požadavku (N2).

V uživatelském rozhraní jsou k dispozici různá nastavení, uvedeme ty zásadní.

- **Nahrávání:** výběr akcí povolených k nahrávání
- **Připojení k prohlížeči:** definování specifického viewportu, typ a informace o připojení (lokální s cestou, vzdálené s IP adresou a portem)
- **Generátor kódu:** přidání řádku importujícího Puppeteer, přidání prázdného řádku mezi bloky kódu, zachytávání chyby
- **Přehrávání:** délka čekání mezi údery kláves, maximální doba čekání na aktuálně neexistující element
- **Backend a interpreter Node.js:** cesta k backendu a interpreteru Node.js

Toto splňuje požadavky (F5)–(F6).

Zbývá nám poslední požadavek, který nebyl vyhodnocen, tím je (N1). Veškerá logika backendové části je postavena nad standardním Puppeteerem. Kód, který je generován, ať už pro účely přehrávání, nebo pouhého nahlédnutí na podobu skriptu, je také kompatibilní se standardním Puppeteerem, proto je požadavek (N1) splněn.

A Fragmenty kódu pro Puppeteer

Zde jsou umístěny části kódu pro Puppeteer, které nejsou považovány za nezbytné, případně jsou příliš dlouhé.

A.1 Zachycení událost z viewportu

```
1 import puppeteer from 'puppeteer'
2
3 const eventsToRecord = ['click', 'dblclick', 'change', 'input',
4                         'select', 'submit', 'scroll', 'copy',
5                         'paste', 'keydown', 'keyup', 'mouseover']
6
7 async function connectEvents(page) {
8   await page.evaluate(viewportEvents, eventsToRecord)
9   await page.evaluateOnNewDocument(viewportEvents, eventsToRecord)
10  await page.exposeFunction('captureViewportEvent',
11                            captureViewportEvent)
12 }
13
14 function viewportEvents(eventsToRecord) {
15   const listener = event =>
16     window.captureViewportEvent({type: event.type})
17
18   eventsToRecord.forEach(eventName =>
19     window.addEventListener(eventName, listener))
20 }
21
22 const captureViewportEvent = eventInfo => console.log(eventInfo)
23
24 const browser = await puppeteer.launch({headless: false,
25                                         defaultViewport: null})
26
27 browser.on('targetcreated', async target => {
28   if(target.type() === 'page') {
29     const page = await target.page()
30     await connectEvents(page)
31   }
32 })
33
34 const startingPage = (await browser.pages())[0]
35 await connectEvents(startingPage)
```

Ukázka kódu 24: Zachytávání událostí z viewportu

Puppeteer vždy spouští Chrome s jedním otevřeným tabem, vzhledem k tomu, že se nejedná o explicitně otevřený tab, jeho otevření nezpůsobí vykonání ❹. To je důvodem, proč navíc voláme ❶ na posledním řádku. Všechny ostatní vytvořené taby mají inicializované hlášení událostí pomocí kódu v ❺.

Napojení JavaScriptu Chromu a JavaScriptu, ve kterém běží Puppeteer se provede ve ❸. Druhý parametr udává metodu ke zveřejnění do window objektu Chromu, první parametr je jméno, pod kterým je metoda dostupná uvnitř okna prohlížeče.

V ❺ se vypisují ohlášené události ze ❹. O provedení těla ❹ z JavaScriptu prohlížeče bychom přišli při navigaci, pokud bychom nezavolali ❷ při inicializaci hlášení.

A.2 Momentálně aktivní tab

```
1  async function getActivePage(browser) {
2    const pages = await browser.pages();
3    for (const p of pages) {
4      const isVisible = await p.evaluate(() => {
5        return document.visibilityState === 'visible'
6      })
7      if(isVisible)
8        return p
9    }
10   throw new Error('No page is currently active.')
11 }
```

Ukázka kódu 25: Metoda, která vrátí momentálně aktivní tab

A.3 Oprávnění „push“

```
1  import puppeteer from 'puppeteer'
2
3  (async () => {
4    const browser = await puppeteer.launch({headless: false})
5    const context = browser.defaultBrowserContext()
6    await context.overridePermissions('https://seznam.cz', ['push'])
7    await browser.close()
8  }) ()
```

Ukázka kódu 26: Pokus o získání oprávnění „push“

Výjimka se objeví na 6. řádku.

A.4 Element podle lokátoru

```
1 async function elementByAttribute(page, attrName, attrValue) {
2   const xpath = `//*[@${attrName} = '${attrValue}']`
3   const found = await page.$x(xpath)
4   if(found.length === 1)
5     return found[0]
6   else if (found.length === 0)
7     throw new Error(`Element not found: ${xpath}`)
8   else {
9     console.log(`Warning Xpath: ${xpath} is ambiguous`)
10    return found[0]
11  }
12 }
13
14 async function elementByLinkText(page, linkText) {
15   const xpath = `//*[text() = '${linkText}']`
16   const found = await page.$x(xpath)
17   if(found.length === 1)
18     return found[0]
19   else if (found.length === 0)
20     throw new Error(`Element not found: ${xpath}`)
21   else {
22     console.log(`Warning Xpath: ${xpath} is ambiguous`)
23     return found[0]
24   }
25 }
```

Ukázka kódu 27: Metody převádějící vybrané lokátory na XPath s návratovou hodnotu odpovídajících elementů

A.5 Úspěšné čtení ze schránky i zápis do schránky

Fungující příklad vyrobíme rozšířením uk. k. 29 z A.6 přidáním uk. k. 28 za 4. řádek.

```
1   const cdpClient = await page0.target().createCDPSession()
2   await cdpClient.send('Browser.grantPermissions', {
3     origin: 'https://www.seznam.cz',
4     permissions: ['clipboardReadWrite', 'clipboardSanitizedWrite']
5   })
```

Ukázka kódu 28: Získání oprávnění „clipboardReadWrite“
a „clipboardSanitizedWrite“

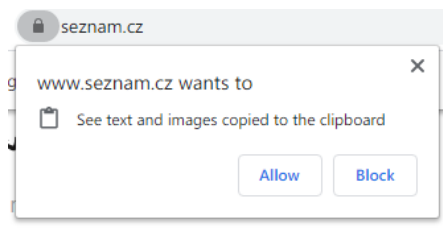
A.6 Práce se stránkou

```
1 (async () => {
2   const browser = await puppeteer.launch({headless: false})
3   const pages = await browser.pages()
4   let page0 = pages[0]
5
6   await Promise.all([
7     page0.waitForNavigation(),
8     page0.goto('https://www.seznam.cz/')
9   ])
10
11   await page0.waitForXPath(`//*[@name = 'username']`)
12   await elementByAttribute(page0, 'name', 'username')
13     .then(el => el.type('uzivatel', { "delay": 100 }))
14
15
16   await elementByAttribute(page0, 'name', 'username')
17     .then(el => el.evaluate((el) => {
18       el.setSelectionRange(0, 5, 'backward') // 'uziv'
19     })))
20
21   ❶ await page0.evaluate(async () => {
22     const currSelection = window.getSelection().toString()
23     await navigator.clipboard.writeText(currSelection)
24   })
25
26   await page0.waitForSelector(`.search-form__input`)
27   await page0.click(`.search-form__input`)
28
29   await page0.$eval(`.search-form__input`, async el => {
30     el.value = await navigator.clipboard.readText()
31   })
32 }) ()
```

Ukázka kódu 29: Pokus o zápis a čtení ze stránky bez jakéhokoliv oprávnění

Kód z uk. k. 29 je nutné doplnit o metodu `elementByAttribute`, jejíž implementace je uvedena v A.4.

Zatímco zápis do schránky proběhne úspěšně, při čtení se zobrazí výzva k udělení/zamítnutí oprávnění.



Obrázek 69: Výzva pro udělení/zamítnutí oprávnění pro čtení schránky

A.7 Pokus o získání oprávnění pro práci se schránkou

Zkusíme-li rozšířit uk. k. 29 z A.6 přidáním níže uvedené uk. k. 30 za 2. řádek, program se ukončí na řádku označeném ❷ (uk. k. 29) a ohlásí výjimku „Error: Evaluation failed: DOMException: Write permission denied.“.

```
1 const ctx = browser.defaultBrowserContext()
2 await ctx.overridePermissions(
3   'https://www.seznam.cz',
4   ['clipboard-read', 'clipboard-write']
5 )
```

Ukázka kódu 30: Pokus o získání oprávnění pro čtení a zápis schránky (nepřímý způsob)

Stejný dopad bude mít i následující rozšíření, které je ale nutné přidat až za 4. řádek.

```
1 const cdpClient = await page0.target().createCDPSession()
2 await cdpClient.send('Browser.grantPermissions', {
3   origin: 'https://www.seznam.cz',
4   permissions: ['clipboardReadWrite']
5 })
```

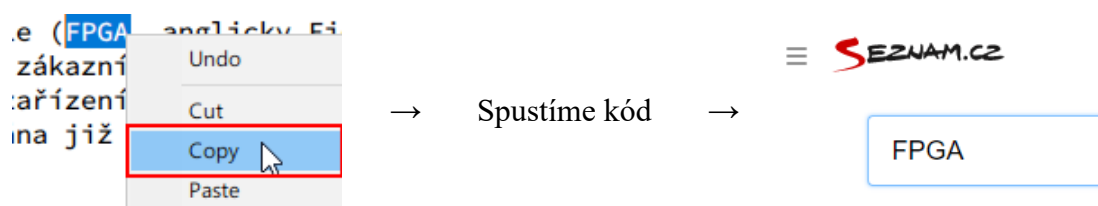
Ukázka kódu 31: Pokus o získání oprávnění pro čtení a zápis schránky (nativní způsob)

A.8 Čtení ze schránky s CDP oprávněním „clipboardReadWrite“

```
1 (async () => {
2   const browser = await puppeteer.launch({headless: false})
3
4   /*
5    const ctx = browser.defaultBrowserContext()
6    await ctx.overridePermissions(
7      'https://www.seznam.cz',
8      ['clipboard-read', 'clipboard-write']
9    )
10  */
11
12  const pages = await browser.pages()
13  let page0 = pages[0]
14
15  /* Tento úsek kódu je ekvivalentní s~výše zakomentovaným
16     fragmentem */
17  const cdpClient = await page0.target().createCDPSession()
18  await cdpClient.send('Browser.grantPermissions', {
19    origin: 'https://www.seznam.cz',
20    permissions: ['clipboardReadWrite']
21  })
22
23  await Promise.all([
24    page0.waitForNavigation(),
25    page0.goto('https://www.seznam.cz/')
26  ])
27
28  await page0.waitForSelector('.search-form__input')
29  await page0.click('.search-form__input')
30
31  await page0.$eval('.search-form__input', async el => {
32    el.value = await navigator.clipboard.readText()
33  })
34 }) ()
```

Ukázka kódu 32: Test čtení ze schránky s CDP oprávněním „clipboardReadWrite“

Výše uvedený kód můžeme využít k ověření funkčnosti čtení ze schránky takto:



Obrázek 70: Využití uk. k. 32 k ověření funkčnosti čtení ze schránky

A.9 Využití NetMQ a ZeroMQ.js

Následují dvě ukázky, jedna pro C#, druhá pro JavaScript s Node.js. Při spuštění obou ukázek se navzájem pošlou verze běhového prostředí. Povšimněme si, že nepotřebujeme message broker, a navíc můžeme odesílat přímo textové řetězce.

```
1 using System;
2 using NetMQ;
3 using NetMQ.Sockets;
4
5 class Program {
6     static void Main(string[] args) {
7         using (PairSocket pair =
8             new PairSocket("@tcp://127.0.0.1:3333")) {
9
10             pair.SendFrame(".NET Framework " + typeof(string).Assembly.
11                 ImageRuntimeVersion);
12
13             string nodejsVersion = pair.ReceiveFrameString();
14             Console.WriteLine(nodejsVersion);
15         }
16     }
17 }
```

(a) C#

```
1 import zmq from 'zeromq'
2
3 let sock = new zmq.Pair
4 sock.connect('tcp://127.0.0.1:3333')
5
6 await sock.send('Node.js ' + process.version)
7 const dotnetVersion = (await sock.receive()).toString('utf-8')
8 console.log(dotnetVersion)
```

(b) JavaScript

Ukázka kódu 33: Demo zobrazující použití NetMQ a ZeroMQ.js

Slovníček vybraných pojmů

Automatizace prohlížeče Proces bezobslužného ovládání prohlížeče.

Chrome DevTools Protocol Protokol umožňující externím nástrojům ovládat a debugovat Chrome [14].

DevTools (Developer Tools) Sada nástrojů vestavěná ve Chromu umožňující debugování a diagnostiku stránek [13].

Electron Framework pro vytváření nativních aplikací pro OS Windows/macOS/Linux pomocí JavaScriptu, HTML, CSS a dalších webových technologií [18].

Extension API (Extension Application Programming Interface) Rozhraní pro programování rozšíření do Chromu, do značné míry kompatibilní s Firefoxem [9, 12, 75].

Firefox Remote Protocol Protokol využívaný ve Firefoxu kompatibilní s CDP. Implementována je pouze podmnožina CDP [56, 57].

Headless Režim spuštění prohlížeče bez grafického uživatelského rozhraní [8].

Headless Recorder (dříve Puppeteer Recorder) Nahrávač pro Puppeteer implementovaný s využitím Extension API [24].

Identifikátor elementu Libovolný způsob (selektor, lokátor, XPath, ...) určení elementu na stránce.

Katalon Recorder Konkurenční řešení k nahrávání a přehrávání akcí. Fork Selenia IDE [39].

Kontext prohlížeče Prostředí jedné i více stránek prohlížeče. Obvykle se sdíleným nastavením, např. neukládat historii [10, 66].

Lokátor Rozšířený identifikátor, umožňující lépe identifikovat elementy, zejména textovým popiskem elementu. Platí, že lokátory \supset selektory [67].

Message Broker Samostatný program zpracovávající zprávy od odesílatele, které předá adresátovi. Často využívaný pokud obě strany nepoužívají stejný poštovní protokol [74].

Mono Implementace .NET Framework s otevřeným zdrojovým kódem podporující platformy: Linux, macOS, BSD a Windows [35, 65].

Nahrávač (recorder) Nástroj k nahrávání.

Nahrávání (recording) Proces zaznamenávání běžných uživatelských akcí prováděných s prohlížečem.

Navigace Načtení jiné stránky s odlišnou URL [49].

Node.js Běhové prostředí JavaScriptu běžící vně prohlížeče [6].

PhantomJS Headless prohlížeč speciálně navržený tak, aby byl skriptovatelný [45].

Playwright Fork Puppeteeru spravovaný firmou Microsoft [46].

Puppeteer Knihovna pro Node.js poskytující API pro ovládání Chromu. Její funkcionality je závislá na protokolu CDP [49].

Přehrávač (player) Nástroj pro přehrávání.

Přehrávání (playing) Proces automatického vykonávání nahraných akcí v prohlížeči.

Ranorex Recorder Komerční řešení pro automatizaci GUI aplikací a prohlížečů [15].

Selenium Konkurenční sada technologií pro ovládání prohlížečů [5].

Selenium IDE Konkurenční nahrávač a přehrávač pro Selenium [63].

TypeScript Rozšíření JavaScriptu o typy a jejich statickou kontrolu. TypeScript je vyvinutý a spravovaný firmou Microsoft [69].

Viewport Viditelná část stránky prohlížeče [72].

Seznam obrázků

1	Komunikace Selenia s prohlížečem	8
2	Komunikace Puppeteeru s Chromem	10
3	Výběr z lokátorů identifikujících element	18
4	Změna elementu bez ručního přepisování lokátoru	19
5	Zpětné zobrazení identifikovaného elementu	19
6	Stránky MFF UK s nabídkovým elementem	20
7	Explicitní zaznamenání „mouseover“ akce	20
8	Přehrávání s breakpointem na 3. akci	21
9	Porovnání GUI Katalon Recorderu a Selenia IDE	22
10	Porovnání podporovaných jazyků pro export	23
11	Použití Headless Recorderu	25
12	Nastavení Headless Recorderu	26
13	Několik elementů, jejichž změna hodnoty vyvolá událost „change“	32
14	Diagram řešení	38
15	Diagram hlavního okna	39
16	Diagram editace existující nahrávky	40
17	Diagram editace nastavení	41
18	Škálování nastavené na 150%	48
19	Porovnání editačního UI při zapnutém/vypnutém škálování	49
20	Nastavení připojení ke Chromu	51
21	Nastavení cesty k interpreteru Node.js a backendu	52
22	Vytvoření nové nahrávky	52
23	Editační UI pro nahrávky	53
24	Změna jména nahrávky	53
25	Spouštění backendu a Chromu	53
26	Editační UI po spuštění backendu a prohlížeče	54
27	Zapnutí nahrávání akcí	54
28	Nahrání první akce	54
29	Vypnutí nahrávání akcí	55
30	Optimalizace akcí	55
31	Zobrazení kódu pro Puppeteer odpovídající nahraným akcím	56
32	Spuštění přehrávání akcí	56
33	Spuštění přehrávání akcí	57
34	Okno popisující stav spuštěného přehrávání	57
35	Zvýraznění akce, která vyvolala chybu	58
36	Odpojení prohlížeče a vypnutí backendu	58
37	Uložení nahrávky a návrat do seznamu všech uložených nahrávek	59
38	Schéma zapojení počítačů	59

39	Nastavení připojení na „Connect“ s vyplněním IP adresy a portu . . .	60
40	Tlačítko JSON jako součást každé akce	61
41	Data ve formátu JSON popisující nahranou akci „pageUrlChanged“ .	61
42	Modifikace nezpracovaných dat v JSONu	62
43	Přepínače pro výběr akcí ke zpracování	62
44	Zaškrtávací políčka pro výběr a povolení akcí	63
45	Vygenerování kódu jedné akce typu „pageUrlChanged“	63
46	Otevření okna pro aplikaci filtrů	64
47	Aktivace filtru pro akce pocházející z okna prohlížeče a viewportu . .	65
48	Skrytí dvou akcí typu „click“ pomocí filtru	66
49	Vztahy mezi přepínači a políčky	67
50	Seznam dostupných lokátorů pro konkrétní akci typu „click“	67
51	Vlastní lokátor	68
52	Data ve formátu JSON akce typu „click“	68
53	Přidání nového lokátoru	69
54	Formulář CodeGenEditor s komponentou ScintillaNET.Scintilla	76
55	Formulář CodeGenSettingsForms s komponentou PropertyGrid	77
56	Formulář FilterForm	78
57	Formulář JSONEditor s komponentou ScintillaNET.Scintilla	79
58	Stavy AppMode.List a AppMode.Edit formuláře MainForm .	80
59	Formulář NodeJsConfig	81
60	Formulář PlayerForm s komponentou PropertyGrid	82
61	Formulář RecorderSettingsForm s uživatelskou komponentou PuppeteerOptionsUserControl	83
62	Formulář ReplayView ve stavu ukončeného přehrávání s jednou chybou	84
63	Formulář WaitingWindow	84
64	Uživatelská komponenta ActionUserControl pro akci typu „click“	85
65	Uživatelská komponenta EditUserControl jako součást formu- láře MainForm	86
66	Uživatelská komponenta PuppeteerOptionsUserControl . .	88
67	Uživatelská komponenta ThumbnailUserControl zobrazující náhled nahrávky s názvem „Webik“ provedenou na stránce webik. ms.mff.cuni.cz	89
68	Výčtový typ WaitForNavigation jako rozbalovací seznam v pro- perty gridu formuláře PlayerForm	95
69	Výzva pro udělení/zamítnutí oprávnění pro čtení schránky	101
70	Využití uk. k. 32 k ověření funkčnosti čtení ze schránky	102

Seznam tabulek

1	Využitý software a technologie včetně verzí	6
2	Seznam lokátorů podporovaných Seleniem [67]	9
3	Prohlížeče podporované Seleniem [67]	9
4	Několik odpovídajících verzí Chromu a Puppeteeru [50]	10
5	Porovnání čekání Playwrightu a Puppeteeru [47, 50]	14
6	Porovnání data vydání prvních verzí prohlížečů podporujících headless režim [20, 31, 52]	16
7	Orientační porovnání existujících řešení	17
8	Oprávnění pro schránku definované CDP a Puppeteerem	34
9	Skutečný význam oprávnění CDP pro schránku	34
10	Porovnání podporovaných událostí viewportu napříč řešeními	42
11	Porovnání podporovaných událostí okna prohlížeče napříč řešeními	43
12	Porovnání podporovaných identifikátorů elementů napříč řešeními	43
13	Porovnání podporovaných druhů čekání napříč řešeními	44
14	Porovnání podpory generování kódu napříč řešeními	45
15	Porovnání dalších pokročilých funkcí napříč řešeními	46
16	Otestované verze OS Windows rozdělené podle nastavení DPI	47
17	Systémové požadavky backendu a frontendu	50
18	Závislosti backendu a frontendu	50

Seznam ukázek kódu

1	HTML kód tlačítka pro přihlášení	8
2	Screenshot v Node.js kódu	11
3	Node.js kód pro vyhodnocení JavaScriptu prohlížeče	12
4	Node.js kód pro kliknutí na element	12
5	Porovnání kódu Playwrightu a Puppeteeru pro stisk elementu po navigaci	13
6	Akce se specificky identifikovanými elementy	14
7	XPath ekvivalentní identifikátoru elementu na 4. řádku	14
8	Stažení instalátoru Firefoxu	15
9	Uložení screenshotu celé stránky do PDF s využitím PhantomJS . . .	16
10	Zachycení otevření nového tabu	27
11	Otevření nového tabu	27
12	Zachycení uzavření existujícího tabu	28
13	Uzavření existujícího tabu	28
14	Zachycení změny adresy URL tabu	28
15	Nastavení tabu jako aktivního	29
16	Kliknutí na element odpovídající selektoru	30
17	Přejetí myši na element odpovídající selektoru	30
18	Odeslání formuláře pomocí Web API	30
19	Rolování stránky	31
20	Dvojitě kliknutí na element odpovídající selektoru	31
21	Výběr textu	32
22	Přehrávání události „change“	33
23	Přesměrování vzdáleně přístupného portu 9223 na port 9222	60
24	Zachytávání událostí z viewportu	97
25	Metoda, která vrátí momentálně aktivní tab	98
26	Pokus o získání oprávnění „push“	98
27	Metody převádějící vybrané lokátory na XPath s návratovou hodnotu odpovídajících elementů	99
28	Získání oprávnění „clipboardReadWrite“ a „clipboardSanitizedWrite“	99
29	Pokus o zápis a čtení ze schránky bez jakéhokoliv oprávnění	100
30	Pokus o získání oprávnění pro čtení a zápis schránky (nepřímý způsob)	101
31	Pokus o získání oprávnění pro čtení a zápis schránky (nativní způsob)	101
32	Test čtení ze schránky s CDP oprávněním „clipboardReadWrite“ . . .	102
33	Demo zobrazující použití NetMQ a ZeroMQ.js	103

Reference

- [1] .NET API browser | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/?view=netframework-4.8>
- [2] .net - When creating a new GUI, is WPF the preferred choice over Windows Forms? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/57909/when-creating-a-new-gui-is-wpf-the-preferred-choice-over-windows-forms>
- [3] .net - WinForms 4K and 1080p scaling / high DPI? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online]. Dostupné z: <https://stackoverflow.com/questions/49012233/winforms-4k-and-1080p-scaling-high-dpi>
- [4] 16 reasons why to use Selenium IDE in 2019 (and 2 why not) - Automated Visual Testing | Applitools. Automated Visual Testing with Visual AI [online]. Copyright © 2020 Applitools. All Rights Reserved. [cit. 14.11.2020]. Dostupné z: <https://applitools.com/blog/why-selenium-ide-2019>
- [5] About Selenium. SeleniumHQ Browser Automation [online] [cit. 2.5.2020]. Dostupné z: <https://www.selenium.dev/about>
- [6] About | Node.js. [online]. Copyright © OpenJS Foundation. All Rights Reserved. Portions of this site originally [cit. 30.10.2020]. Dostupné z: <https://nodejs.org/en/about>
- [7] Archiving the project: suspending the development · Issue #15344 · ariya/phantomjs · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 7.11.2020]. Dostupné z: <https://github.com/ariya/phantomjs/issues/15344>
- [8] Automated testing with Headless Chrome | Web | Google Developers. Google Developers [online] [cit. 2.5.2020]. Dostupné z: <https://developers.google.com/web/updates/2017/06/headless-karma-mocha-chai>
- [9] Browser Extensions - Mozilla | MDN. [online]. Copyright © 2005 [cit. 23.1.2021]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions>
- [10] Browsing context - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. [online]. Copyright © 2005 [cit. 7.11.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Browsing_context
- [11] Choose your Windows app platform - Windows applications | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform>

- [12] Chrome APIs - Google Chrome. [online] [cit. 2.5.2020]. Dostupné z: https://developer.chrome.com/extensions/api_index
- [13] Chrome DevTools | Google Developers. Google Developers [online] [cit. 2.5.2020]. Dostupné z: <https://developers.google.com/web/tools/chrome-devtools>
- [14] Chrome DevTools Protocol [online] [cit. 2.11.2020]. Dostupné z: <https://chromedevtools.github.io/devtools-protocol>
- [15] Company Profile | About Ranorex. Test Automation for GUI Testing | Ranorex [online]. Copyright © 2020 Ranorex GmbH. All Rights Reserved [cit. 9.12.2020]. Dostupné z: <https://www.ranorex.com/company>
- [16] Compatibility | Mono. Home | Mono [online]. Copyright © 2020 Mono Project [cit. 19.12.2020]. Dostupné z: <https://www.mono-project.com/docs/about-mono/compatibility>
- [17] Ecosystem. SeleniumHQ Browser Automation [online] [cit. 31.10.2020]. Dostupné z: <https://www.selenium.dev/ecosystem>
- [18] Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS. [online] [cit. 14.11.2020]. Dostupné z: <https://www.electronjs.org>
- [19] Element - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Element>
- [20] Firefox Release Calendar - MozillaWiki. [online] [cit. 8.11.2020]. Dostupné z: https://wiki.mozilla.org/Release_Management/Calendar
- [21] Frequently asked questions - Firefox add-on technology is modernizing | Firefox Help [online]. Copyright ©1998 [cit. 14.11.2020]. Dostupné z: <https://support.mozilla.org/en-US/kb/frequently-asked-questions-firefox-addon>
- [22] Getting Started with Headless Chrome | Web | Google Developers. Google Developers [online] [cit. 7.11.2020]. Dostupné z: <https://developers.google.com/web/updates/2017/04/headless-chrome>
- [23] GitHub - SeleniumHQ/selenium-ide at v3. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 14.11.2020]. Dostupné z: <https://github.com/SeleniumHQ/selenium-ide/tree/v3>
- [24] GitHub - checkly/headless-recorder: Headless recorder is a Chrome extension that records your browser interactions and generates a Puppeteer or Playwright script.. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 30.10.2020]. Dostupné z: <https://github.com/checkly/headless-recorder>

- [25] GitHub - jacobslusser/ScintillaNET: A Windows Forms control, wrapper, and bindings for the Scintilla text editor.. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 14.1.2021]. Dostupné z: <https://github.com/jacobslusser/ScintillaNET>
- [26] GitHub - Kubikola/puppeteer-automation: Puppeteer automation (work in progress). GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 28.1.2021]. Dostupné z: <https://github.com/Kubikola/puppeteer-automation>
- [27] GitHub - microsoft/playwright: Node.js library to automate Chromium, Firefox and WebKit with a single API. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright>
- [28] GitHub - zeromq/clrzmq4: ZeroMQ C# namespace (.NET and mono, Windows, Linux and MacOSX, x86 and amd64). GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/clrzmq4>
- [29] GitHub - zeromq/netmq: A 100% native C# implementation of ZeroMQ for .NET. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/netmq>
- [30] GitHub - zeromq/zeromq.js: Node.js bindings to the ØMQ library. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/zeromq.js>
- [31] Google Chrome version history - Wikipedia. [online] [cit. 8.11.2020]. Dostupné z: https://en.wikipedia.org/wiki/Google_Chrome_version_history
- [32] google chrome - Using Chromium Remote Debugging from External Device - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 4.1.2021]. Dostupné z: <https://stackoverflow.com/questions/18506233/using-chromium-remote-debugging-from-external-device>
- [33] HTMLElement - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>
- [34] HTMLFormElement.submit() - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement/submit>
- [35] Home | Mono. Home | Mono [online]. Copyright © 2020 Mono Project [cit. 18.12.2020]. Dostupné z: <https://www.mono-project.com>

- [36] Index | Node.js v15.4.0 Documentation. [online] [cit. 19.12.2020]. Dostupné z: <https://nodejs.org/api>
- [37] Index. SeleniumHQ Browser Automation [online] [cit. 3.11.2020]. Dostupné z: <https://www.selenium.dev/selenium/docs/api/javascript>
- [38] Is WPF replacement of WinForms? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/4852407/is-wpf-replacement-of-winforms/4852581>
- [39] Katalon Recorder (Selenium tests generator) - Internetový obchod Chrome. [online] [cit. 2.5.2020]. Dostupné z: <https://chrome.google.com/webstore/detail/katalon-recorder-selenium/ljdobmmdgdljniojadhoplhkpialdid>
- [40] Katalon Recorder vs Katalon Studio | Katalon Docs. Redirecting... [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 17.11.2020]. Dostupné z: <https://docs.katalon.com/katalon-recorder/docs/katalon-recorder-vs-katalon-studio.html>
- [41] Katalon Solution. Katalon Solution [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 16.11.2020]. Dostupné z: <https://www.katalon.com/katalon-recorder-ide>
- [42] Katalon Solution. Katalon Solution [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 16.11.2020]. Dostupné z: <https://www.katalon.com/pricing>
- [43] Katalon Studio - Wikipedia. [online] [cit. 16.11.2020]. Dostupné z: https://en.wikipedia.org/wiki/Katalon_Studio
- [44] NavigatorID.appName - Web APIs | MDN. [online]. Copyright © 2005 [cit. 3.11.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/NavigatorID/appName>
- [45] PhantomJS - Scriptable Headless Browser. PhantomJS - Scriptable Headless Browser [online]. Copyright © 2010 [cit. 7.11.2020]. Dostupné z: <https://phantomjs.org>
- [46] Playwright. Playwright [online] [cit. 5.11.2020]. Dostupné z: <https://playwright.dev>
- [47] playwright/api.md at master · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright/blob/master/docs/api.md>
- [48] playwright/browser_patches at master · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: https://github.com/microsoft/playwright/tree/master/browser_patches

- [49] Puppeteer [online] [cit. 2.11.2020]. Dostupné z: <https://pptr.dev>
- [50] puppeteer/api.md at main · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 3.11.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/blob/main/docs/api.md>
- [51] puppeteer/api.md at v5.5.0 · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 16.1.2021]. Dostupné z: <https://github.com/puppeteer/puppeteer/blob/v5.5.0/docs/api.md#pagewaitfornavigationoptions>
- [52] Release 1.0.0 · ariya/phantomjs · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 7.11.2020]. Dostupné z: <https://github.com/ariya/phantomjs/releases/tag/1.0.0>
- [53] Releases · SeleniumHQ/selenium-ide · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 14.11.2020]. Dostupné z: <https://github.com/SeleniumHQ/selenium-ide/releases>
- [54] Releases · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright/releases?after=v0.13.0>
- [55] Releases · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 2.11.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/releases?after=v0.12.0>
- [56] Remote - MozillaWiki. [online] [cit. 2.11.2020]. Dostupné z: <https://wiki.mozilla.org/Remote>
- [57] Remote Protocol — Firefox Source Docs documentation. Firefox Source Tree Documentation — Firefox Source Docs documentation [online] [cit. 2.11.2020]. Dostupné z: <https://firefox-source-docs.mozilla.org/remote/index.html>
- [58] Request: browser.currentPage() or similar way to access Pages · Issue #443 · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 12.12.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/issues/443>
- [59] Selectors Level 3. World Wide Web Consortium (W3C) [online]. Copyright © 2018 [cit. 9.11.2020]. Dostupné z: <https://www.w3.org/TR/selectors-3>
- [60] Selectors Level 4. World Wide Web Consortium (W3C) [online]. Copyright © 2018 [cit. 9.11.2020]. Dostupné z: <https://www.w3.org/TR/selectors-4>

- [61] Selenium (software) - Wikipedia. [online] [cit. 2.11.2020]. Dostupné z: [https://en.wikipedia.org/wiki/Selenium_\(software\)](https://en.wikipedia.org/wiki/Selenium_(software))
- [62] Selenium History. SeleniumHQ Browser Automation [online] [cit. 14.11.2020]. Dostupné z: <https://www.selenium.dev/history>
- [63] Selenium IDE · Open source record and playback test automation for the web. SeleniumHQ Browser Automation [online]. Copyright © 2019 Software Freedom Conservancy [cit. 30.10.2020]. Dostupné z: <https://www.selenium.dev/selenium-ide>
- [64] SetProcessDpiAwareness function (shellscalingapi.h) - Win32 apps | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 13.2.2021]. Dostupné z: <https://docs.microsoft.com/en-us/windows/win32/api/shellscalingapi/nf-shellscalingapi-setprocessdpiawareness>
- [65] Supported Platforms | Mono. Home | Mono [online]. Copyright © 2021 Mono Project [cit. 21.1.2021]. Dostupné z: <https://www.mono-project.com/docs/about-mono/supported-platforms>
- [66] Terminology - HTML5. W3C on GitHub [online] [cit. 7.11.2020]. Dostupné z: <https://w3c.github.io/html-reference/terminology.html>
- [67] The Selenium Browser Automation Project :: Documentation for Selenium [online] [cit. 2.11.2020]. Dostupné z: <https://www.selenium.dev/documentation/en>
- [68] The history of C# - C# Guide | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 18.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>
- [69] TypeScript: Typed JavaScript at Any Scale.. TypeScript: Typed JavaScript at Any Scale. [online]. Copyright © 2012 [cit. 17.12.2020]. Dostupné z: <https://www.typescriptlang.org>
- [70] UWP Documentation - UWP app developer - UWP applications | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/windows/uwp>
- [71] Using Headless Mode in Firefox - Mozilla Hacks - the Web developer blog. Home - Mozilla Hacks - the Web developer blog [online] [cit. 7.11.2020]. Dostupné z: <https://hacks.mozilla.org/2017/12/using-headless-mode-in-firefox>
- [72] Viewport concepts - CSS: Cascading Style Sheets | MDN. [online]. Copyright © 2005 [cit. 7.5.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport_concepts
- [73] Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio. [online] [cit. 18.12.2020]. Dostupné z: <https://visualstudio.microsoft.com>

- [74] What are Message Brokers? | IBM. [online]. Copyright © Copyright IBM Corporation 2020 [cit. 28.12.2020]. Dostupné z: <https://www.ibm.com/cloud/learn/message-brokers>
- [75] What are extensions? - Google Chrome. [online] [cit. 2.5.2020]. Dostupné z: <https://developer.chrome.com/extensions>
- [76] What is WPF? - Visual Studio | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019>
- [77] win: block running on EOL Windows versions by joaocgreis · Pull Request #31954 · nodejs/node · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 28.1.2021]. Dostupné z: <https://github.com/nodejs/node/pull/31954>
- [78] Window - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Window>
- [79] Window: load event - Web APIs | MDN. [online]. Copyright © 2005 [cit. 5.11.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event
- [80] winforms - Will Windows Forms be deprecated in favor of WPF? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/913417/will-windows-forms-be-deprecated-in-favor-of-wpf>
- [81] winforms - Windows Forms Dead. Long life to WPF - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/2632118/windows-forms-dead-long-life-to-wpf>
- [82] wpf - Are Windows Forms old tech? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/1916510/are-windows-forms-old-tech>
- [83] ZeroMQ [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org>
- [84] ZeroMQ | C# [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org/languages/csharp>
- [85] ZeroMQ | Get started [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org/get-started>